ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD CARLOS III DE MADRID

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

ESTUDIO DE VIABILIDAD DE DROOLS-GUVNOR PARA SU INTEGRACIÓN EN EL SIMULADOR EMPRESARIAL SIMBA

Autor: **Diego Jesús Vázquez García**
Tutor: **Fernando Fernández Rebollo**

febrero de 2012

Agradecimientos

En el año 2001 comencé esta aventura que, espero, esté a punto de terminar ahora. En estos años muchos han sido los compañeros de viaje sin los cuales no hubiese sido posible llegar hasta aquí. Sería imposible recogerlos a todos aquí.

Sí que me gustaría recordar a aquellas personas más importantes en este camino, hacer mención a Isabel y Pepe por su apoyo, paciencia y respeto a mi cabezonería, a Diana por su confianza infinita, a Fernando por compartir su talento, a Julio por enseñarme que no sabía nada, a José Alberto por enseñarme que después del invierno viene la primavera y a tantos otros, Pablo, Enrique, David, José Antonio, etc. De todo corazón gracias a todos

Por supuesto no podría dejar fuera a otro Fernando, mi tutor, sin el cual entregar este trabajo hubiese sido completamente imposible, gracias por su interés, flexibilidad y paciencia.

Contenidos

Índice de figuras

Capítulo 1

Introducción

Habitualmente los sistemas informáticos se pueden dividir en tres elementos o capas principales: presentación de la información, datos y lógica. La capa de presentación de la información se encarga de presentar la información de forma adecuada; la capa de datos se encarga de saber qué información procesa la aplicación y dónde se encuentra dicha información; por último, la capa de lógica (habitualmente llamada lógica de negocio), se encarga de tomar decisiones y realizar operaciones en función de los datos que maneja la aplicación para conseguir la información que se desea representar. En la mayoría de las aplicaciones, la capa de lógica suele ser las más importante dentro de la aplicación.

Para el desarrollo de los sistemas informáticos se suelen utilizar lenguajes de programación, que son un conjunto de instrucciones escritas de tal forma que un ordenador pueda interpretarlas. Existen múltiples clasificaciones de los lenguajes de programación, los más comúnmente utilizados son los denominados lenguajes imperativos. Estos lenguajes se caracterizan porque sus instrucciones son órdenes y algunos de los más populares son Java o C.

Cuando nos enfrentamos a estructuras lógicas especialmente complejas, utilizar un lenguaje imperativo puede tener consecuencias negativas. Se puede provocar un progresivo empeoramiento de la calidad del programa; además suele dificultar la verificación del problema que solventa ya que dificulta la interpretación del comportamiento del sistema. Una de las razones por las que surgen estos problemas, es porque unido a la complejidad de la solución que se trata de aplicar, las personas que conocen el lenguaje de programación no son las personas que conocen cómo se debe solucionar el problema para el que se está desarrollando el sistema. Las partes de un programa que se consideran muy complejas y por ello difícilmente interpretables, se suelen denominar de forma peyorativa *código spaguetti*.

Una de las posibilidades para facilitar la tarea de aplicar lógica de decisión de una manera más sencilla y eficiente consiste en emplear herramientas como los *motores de reglas*. Son muy útiles a la hora de construir la lógica de un sistema informático, sobre todo en lo referente a tomar decisiones. Para poder utilizar un motor de reglas es necesario que las estructuras lógicas se encuentren definidas mediante reglas. Las reglas son una forma de

representar decisiones de la forma *SI... ENTONCES...*. Gracias a las reglas que son muy sencillas de crear, podemos utilizar estructuras de decisión fáciles de interpretar, que de emplearse lenguajes imperativos para construirlas, serían excesivamente complejas.

Para hacer completamente operativos estos entornos no es suficiente con un elemento que interprete las reglas, además son necesarios una serie de herramientas que nos ayuden a llevar a cabo el desarrollo completo del sistema, tales como un editor donde construir reglas, un repositorio donde guardar las reglas, un compilador que compruebe que las reglas están bien escritas...a los sistemas que incluyen todas estas funcionalidades, se les conoce como *sistemas para la gestión de reglas de negocio*, más conocidos por sus siglas en inglés *BRMS* (Bussines Rules Managment System).

Existe multitud de herramientas construidas para elaborar sistemas basados en reglas. Uno de los *software* más importantes es *drools*(http://www.jboss.org/drools/). Además es una aplicación de código abierto (open source) mantenida principalmente por la comunidad de Jboss (http://www.jboss.org/).

Construir la lógica de una aplicación utilizando reglas, no es sólo interesante por la mejora en la calidad del sistema, también nos permite implicar en el desarrollo a personas con ninguno o muy pocos conocimiento en lenguajes de programación. En ocasiones es imprescindible involucrar en el desarrollo del sistema a personas con muy pocos conocimientos en la tecnología que se va a aplicar, pero que tienen una gran comprensión del dominio del problema. Ésto puede suceder cuando la solución del sistema tiene que estar sometida a una constante evolución.

Existen multitud de entornos donde se pueden utilizar los motores de reglas. Como factor común todos tienen la complejidad de su lógica de negocio, lo que suele derivar en una gran dificultad a la hora de mantener el sistema. Éste puede ser el caso de algunos simuladores.

Los simuladores son herramientas que intentan imitar el comportamiento de un sistema (entendido en el amplio sentido del término), por ejemplo la bolsa. Son herramientas muy útiles porque nos permiten realizar tareas bastante importantes en un entorno controlado. Sin embargo, algunos simuladores son herramientas tremendamente complejas y necesitan reajustarse de manera continuada, por lo que son un grupo perfecto de aplicaciones donde emplear reglas. El caso que acabamos de describir podría pertenecer al simulador empresarial de SIMBA[1].

SIMBA es una simulador cuyo objetivo es ayudar a la formación para la toma de decisiones en un entorno empresarial. Para ello diversos equipos formados por alumnos, a través de SIMBA, participa en un ”juego”donde cada grupo dirige una empresa y toma una serie de decisiones sobre los aspectos de dicha empresa. Con las decisiones que han tomado los usuarios y otras circunstancias (condiciones del mercado, situaciones especiales...) se realiza una simulación donde se trata de determinar cómo se comportarían dichas empresas en el entorno definido.

Un aspecto muy importante de SIMBA, es que además de se capaz de realizar una simulación para determinar la evolución de una empresa, es capaz de determinar un ranking para ordenar las empresas en función de su comportamiento y así poder evaluar perfecetamente las decisiones que han tomado los equipos.

El presente proyecto pretender emplear drools para sustituir alguna de las partes más complejas del simulador SIMBA (en concreto el módulo de arbitraje). De ésta forma dotaremos a éste simulador de las ventajas de emplear un motor de reglas.

Contenido del documento

El documento se estructura de la siguiente forma:

- **Estado del arte**: Incluye un pequeño resumen de los elementos cuyo conocimiento ha sido necesario para poder llevar a cabo el estudio.
- **Motivación y Objetivos**: Una descripción de cuáles han sido las necesidades que se pretender cubrir, así cómo los objetivos que se esperan alcanzar.
- **Memoria del trabajo**: Una descripción detallada del trabajo que se ha realizado.
- **Validación**: Se comprobará la completitud de los objetivos.
- **Gestión del proyecto**: El detalle de cómo se han gestionado las diferentes etapas del estudio.
- **Conclusiones y Trabajos Futuros**: Finalmente una serie de conclusiones personales, así como unas recomendaciones sobre cómo ampliar el trabajo realizado.

Capítulo 2

Estado del arte

A continuación se expondrán de manera resumida los conocimientos y las tecnologías que han sido necesarias para poder llevar a cabo el presente estudio.

2.1. Sistemas basados en conocimiento

Existen una serie de problemas que son especialmente complejos ya que cuando se tratan de resolver no conocemos alguno de los componentes de su solución, o bien no existe una solución definida, es decir la meta que se quiere alcanzar con el programa está por determinar de manera obvia. A estos problemas se les conoce como *problemas semi-estructurados* o *no estructurados*. Una de sus principales característica es que, a priori, no se puede alcanzar una solución única. Cuando estos problemas están ligados a un contexto (por ejemplo el procesamiento del lenguaje natural), podemos utilizar la *ingeniería del conocimiento* para abordarlos. Cuando el problema lo resolvemos aplicando un determinado conocimiento del dominio hablamos de *sistemas basados en el conocimiento*. [2]

Las tareas principales que realiza la ingeniería del conocimiento son:

- Evaluar y reconocer aplicaciones potenciales, diagnosticando las tareas que se pueden llevar a cabo o las que no sea posible resolver.
- Extraer los conocimientos personales de los expertos del dominio sobre el problema a resolver.
- Diseñar, construir y verificar una(s) base(s) de conocimiento, seleccionando las representaciones que mejor se ajustan.
- Diseñar modelos de resolución de problemas.
- Usar heurísticas que puedan ayudar a resolver el problema.
- Elegir la herramienta adecuada para llevar a buen puerto las tareas anteriores.

Cuando construimos un sistema basado en el conocimiento esperamos obtener las siguientes ventajas en el enfoque sobre el problema:

- **Coste reducido:** en las tareas en que puede aplicarse, es más económico que utilizar continuamente a una persona con los conocimientos/experiencia adecuados.
- **Permanencia:** La experiencia es permanente, por contra los expertos pueden olvidar en un momento dado un dato clave.
- **Experiencia múltiple:** El conocimiento de varios especialistas puede estar disponible para trabajar simultánea y continuamente en un problema.
- **Respuestas no subjetivas:** El sistema experto ofrece respuestas sólidas, completas y sin emociones en todo momento.
- **Explicación del razonamiento:** El sistema experto puede explicar clara y detalladamente el razonamiento que conduce a una conclusión.
- **Respuesta rápida:** Algunas situaciones de emergencia pueden exigir respuestas más rápidas que las de una mente humana.

La utilización de la ingeniería del conocimiento lleva aparejada una serie de riesgos que hay que tener en cuenta. Uno de los más importantes es la adquisición del propio conocimiento. Transmitir el conocimiento de un experto a una persona que no lo es puede ser un tema bastante complicado. El ingeniero del conocimiento puede no definir correctamente al experto, el experto puede olvidar cosas o el ingeniero no encontrar los métodos adecuados, para conseguir la información que necesita. Además, en ocasiones es difícil distinguir que parte de todo el conocimiento que tiene el experto es realmente efectiva para resolver el problema. Una vez obtenido el conocimiento, es necesario encontrar una estructura en la que poder representar el conocimiento de manera que sea un ordenador el que pueda procesar dicha información. Otro de los grandes riesgos es generar las inferencias que procesen la información útil en el contexto de un caso específico.

Como podemos ver en la figura 2.1, la estructura básica de un sistema basado en el conocimiento está formada por la *base de conocimiento*, la *base de hechos* y *el motor de inferencia*. La base de conocimiento es la representación del conocimiento que se ha extraído del experto humano; la base de hechos está formada por datos bajo los que se plantea el problema; y el motor de inferencia, es el responsable de aplicar la base de conocimientos a la base de hechos para encontrar la solución al problema. Puesto que los sistemas no están aislados y necesitan comunicarse con otros sistemas o los usuarios, una parte muy importante son las unidades de entrada/salida, que realizan esta labor.

Existen multitud de problemas donde la aplicación de sistemas basados en el conocimiento es un completo éxito. Algunos de los campos más significativos son: medicina, ayudando a la evaluación de diagnósticos, asesoramiento farmacológico...(`http://www.cfnavarra.es/salud/anales/textos/vol25/n3/colab.html`); geología, interpretando los temblores de tierra... (`http://www.oilproduction.net/files/repsol_ypf_valhondo.pdf`); meteorología, predicciones pluviométricas, consolidación de información... (`http://www.meteo.unican.es/`); empresariales (`http://newkmanager.blogspot.com/2006/11/pldoras-gestin-del-conocimiento-en.html`), apoyo a la

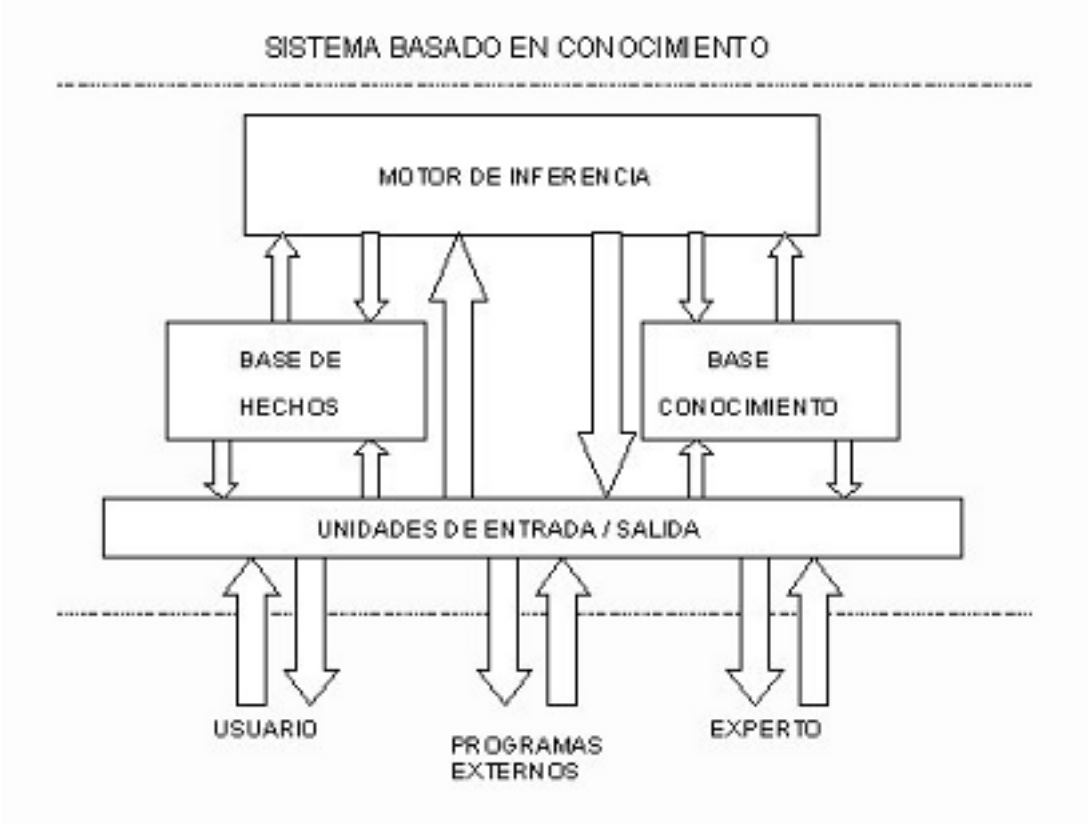


Figura 2.1: Estructura básica de un sistema del conocimiento y sus flujos de interacción [3].

toma de decisiones... y un largo etcétera de campos y problemas donde los sistemas basados en el conocimiento han sido un éxito de aplicación.

En los entornos financieros, los usos más habituales de los sistemas basados en el conocimiento son: planificación financiera, medir el impacto de las decisiones que se toman sobre la empresa; evaluación de riesgos, determinar el balance existente entre beneficios y riesgos; previsiones, predecir cómo será el futuro; testear procesos de modelo de negocio, establecer los pasos de un proceso de negocio de manera fiable; aprendizaje interactivo, emplearlo para formar nuevos empleados en un entorno seguro; entre otros usos.

Los sistemas basados en el conocimiento, por tanto, trabajan en problemas que resuelven personas que conocen en profundidad el dominio del problema. Para construir un método de solución del problema propuesto es necesario trabajar con los expertos y extraer el conocimiento sobre cómo se resuelve para posteriormente elaborar un sistema que pueda representar ese conocimiento y ser empleado posteriormente.

2.2. Motores de reglas

Para tener claro qué es un motor de reglas es necesario conocer lo que es una regla de negocio. Las *reglas de negocio* son una manera sencilla de poder representar conocimiento, por ejemplo ”si hay nubes entonces no se verá el sol”. Las reglas de negocio pueden ser empleadas en las aplicaciones para representar la lógica de las mismas.

Dentro de una regla siempre existen dos partes, antecedente y consecuente. Los antecedentes (o parte izquierda de la regla o LHS por sus siglas en ingles) son un conjunto de condiciones booleanas. Cada condición se suele llamar patrón. El consecuente (o parte derecha de la regla o RHS), es la consecuencia de que el antecedente sea cierto.

En un motor de reglas existen tres elementos imprescindibles (figura 2.2): la *base de hechos*, el *motor de inferencia* y *el conjunto de reglas*. Éstos elementos se pueden identificar con los de los sistemas basados en el conocimiento vistos anteriormente.

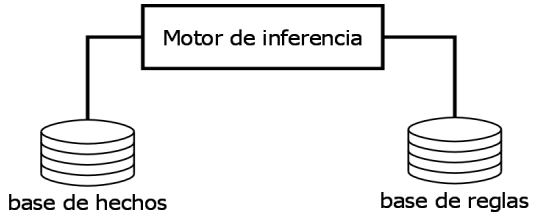


Figura 2.2: Arquitectura de un motor de reglas.

La base de hechos son los datos que conforman el estado del sistema. Es decir los datos a partir de los cuales hay que resolver el problema. El conjunto de reglas, son las reglas que encapsulan el conocimiento necesario para resolver el problema. Por último el motor de inferencia es el responsable de determinar las reglas que hay que ejecutar para resolver el problema.

Los datos de la base de hechos serán los que hagan que los antecedentes de las reglas se cumplan o no. Los consecuentes pueden modificar la base de hechos (ya sea añadiendo nuevos hechos, modificándolos o eliminándolos).

Para que el motor de inferencia y las reglas puedan interpretar correctamente los datos de la base de hechos es necesario que éstos se ajusten a un modelo que los represente. A este modelo se le denomina de muchas maneras, como ontología o modelo de dominio.

El funcionamiento básico del motor de inferencia, consiste en determinar qué reglas se pueden activar (son ciertas todas las condiciones de su antecedente) para el conjunto de datos de la base de hechos. Si existe más de una regla que se puede activar, se produce lo que se llama colisión o conflicto, puesto que sólo se puede ejecutar una regla a la vez. Para determinar cuál es la regla que se puede ejecutar es necesario aplicar algún criterio; entre los más comunes están: la mas reciente en añadirse al sistema, la que más patrones necesite

para activarse (la más específica) o establecer algún tipo de prioridad. Una vez que se ha determinado la regla que se debe ejecutar se evalúa su consecuente, llevando a cabo las operaciones que éste indique. Una cosa a tener en cuenta es que los cambios no se realizan sobre la base de hechos en sí, sino lo que se denomina la memoria de trabajo, que es un elemento que sólo existe dentro del motor de inferencia. El proceso descrito se realiza de manera iterativa hasta que, o bien no existe ninguna regla que se pueda ejecutar, o alguna regla tiene en su consecuente la condición de parada, denominada habitualmente *halt*. Ésta es una descripción muy simple de cómo funciona el motor de inferencia.

Las reglas suelen llevar además una serie de información extra que si bien no es imprescindible, sí ayuda a interpretar la regla (o bien al motor, o bien para facilitar su interpretación a las personas), en ocasiones modificando el comportamiento de la regla. Los más comunes (empleados en diversos motores de reglas), suelen ser: el título de la regla, la descripción de la regla y la prioridad (que suele ser utilizada para resolver el conjunto de colisión). A ésta información se suele denominar meta-información.

Un aspecto muy importante sobre el funcionamiento del motor de inferencia es la manera en que éste determina qué reglas se activan en un momento dado. Si tuviésemos que comparar cada regla con todos los hechos y ver cuáles se activan, veríamos que es un proceso altamente ineficiente. Por ello es muy habitual que los motores de reglas implementen el algoritmo *Rete* [4]. Este algoritmo, crea un árbol donde cada nodo es un patrón de una regla, y las hojas son las reglas. Cuando todos los nodos están activos (todos los patrones son verdaderos) entonces es cuando la regla se puede activar. Este algoritmo se basa en que los patrones se repiten en muchas reglas y en que la activación de una regla que introduzca cambios en el conjunto de hechos no afectará a muchos nodos de la siguiente interacción. Un ejemplo de cómo resulta la estructura del árbol, la tenemos en la figura 2.3.

Aunque todas las aplicaciones incluyen una importante lógica de negocio que se puede representar mediante el uso de reglas de negocio, no siempre es práctico incorporar un motor de reglas en nuestros sistemas. Los casos donde aporta una ventaja significativa frente a un enfoque basado en implementar la lógica de negocio con un lenguaje imperativo, son:

- **El resultado de utilizar un enfoque imperativo es demasiado confuso:** el problema a resolver no es especialmente complejo, sin embargo su implementación sí que es muy compleja, implica replicar estructuras de programación (o utilizar estructuras muy complejas para evitarlo) o genera un código que dificulta mucho la legibilidad.
- **El problema es muy complejo o de difícil compresión:** En ocasiones implementar una lógica de negocio suele ser muy complejo debido a que las personas que se encargan de desarrollar no comprenden el dominio del problema y no saben interpretar los resultados.
- **La lógica de negocio sufre cambios continuamente:** La lógica de la aplicación o una parte identificable de la misma, sufre cambios de manera continua, que sólo la afectan a ella. Es decir la lógica de la aplicación tiene un ciclo de vida distinto al del resto de la aplicación.

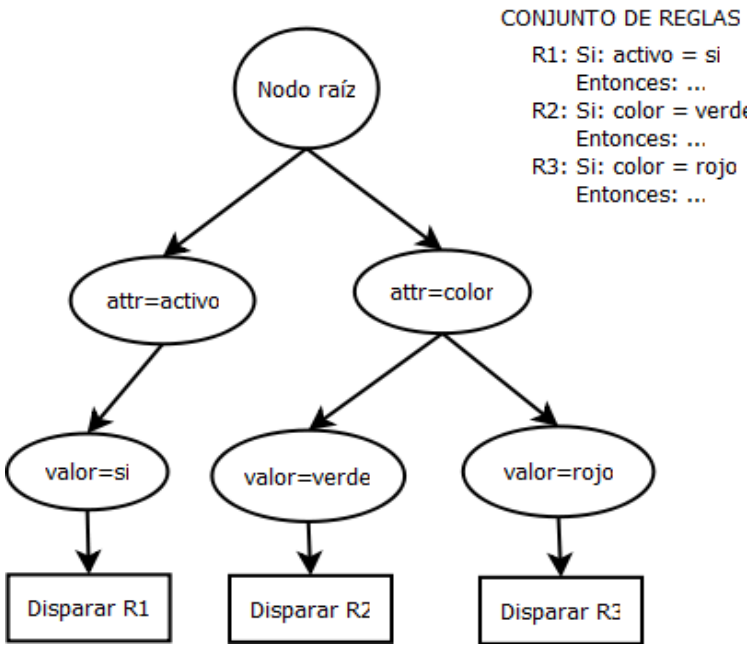


Figura 2.3: Ejemplo simple de árbol generado con el algoritmo Rete. En este ejemplo están definidas tres reglas (R1,R2 y R3), estas tres reglas tienen condiciones diferentes en base a dos atributos (activo y color), las consecuencias de la regla no son importantes para determinar el árbol Rete. La raíz del árbol es un nodo vacío, el primer conjunto de hijos se evalúa que atributo es el que se ha disparado. En el segundo nivel de hijos se evalúan los valores de los atributos, estos se construyen a partir de los patrones que forman parte de la regla, por último las hojas son los consecuentes de las reglas. Una vez construído el árbol es sencillo evaluar las reglas ante cambios en la memoria de trabajo.

- **Es necesario involucrar a expertos del dominio, que no poseen conocimientos técnicos:** En ocasiones la solución final debe ser desarrollada por personas que conocen muy bien el problema, pero que no tienen la capacidad técnica para desarrollar un programa. Las reglas al ser intuitivas permiten a estas personas poder editarlas con un mínimo de formación.

Estos problemas se suelen plantear en ocasiones de manera conjunta. Si nos preguntásemos por qué es importante implementar el módulo de arbitraje de SIMBA (Simulation in Business Administration) con un motor de reglas, principalmente encajaría en el segundo punto. El problema de decidir cómo se ve afectada una empresa en un entorno de mercado en base a sus decisiones (y las de sus competidores) no está completamente definido. Aún siguiendo las especificaciones de los expertos, podemos cometer errores porque el problema es muy complejo, incluso porque los mismos expertos se equivocan. Ésto tiene como consecuencia que la lógica de arbitraje está sujeta a cambios de manera continuada, y además, supone que es necesario conseguir que los expertos del dominio se involucren de manera activa en la construcción del módulo.

Las principales ventajas que nos aportan los motores de reglas son:

- **Fáciles de comprender:** en general las reglas son mucho más fáciles de comprender que un fragmento de código (escrito en lenguaje imperativo) que trate de realizar la misma función.
- **Facilitan la evolución del sistema:** es muy fácil añadir/modificar/eliminar reglas, sin que afecte al comportamiento de los demás componentes del sistema.
- **Flexibilidad:** una misma aplicación puede tener diferentes conjuntos de reglas en su vida, de tal manera que si nuestra lógica es muy compleja o es reescrita a menudo, es mucho más sencillo reescribir un conjunto de reglas, que en general reescribir la lógica de un aplicativo.
- **Reusabilidad:** al ser las reglas independientes de la aplicación se pueden utilizar en diferentes aplicativos (aunque los aplicativos tendrán que tener similar contexto).
- **Mantener un ciclo de vida independiente:** al utilizar un motor de reglas independizamos el ciclo de vida del desarrollo de las reglas del aplicativo. Así podemos desarrollar y probar nuestras reglas al margen de la aplicación y sólo incorporarlas a la aplicación cuando sea necesario. En caso contrario, cada cambio en la lógica supondría pasar todo el proceso de creación de versión nueva del sistema.

Incorporar un motor de reglas a nuestro sistema puede ser bastante costoso, por lo que no siempre es una buena idea incluirlo. Algunas situaciones donde su uso se desaconseja son:

- Si el sistema es demasiado pequeño(sobre 20 reglas). En este caso el coste de integrar el motor puede ser mayor que el beneficio que se obtendría.
- Si la lógica de la aplicación está perfectamente definida y va cambiar poco, o los cambios que se introduciría son pequeños y/o definibles.
- Si las reglas resultan ser excesivamente simples.
- Si el rendimiento de la aplicación es una de las máxima preocupaciones.
- Si una vez que el sistema sea estable, no va sufrir más modificaciones.

Referente al rendimiento, aunque un motor de reglas suponga una perdida de rendimiento respecto a implementar la lógica dentro del sistema, no es tampoco un handicap demasiado grande. Incluso se pueden utilizar motores de reglas en sistemas que requieran respuestas en tiempo real. En problemas con lógica de negocio especialmente complejo donde es muy difícil llevar a cabo una implementación eficiente del sistema, los motores de reglas pueden obtener un rendimiento mejor incluso que un desarrollo llevado a cabo en un lenguaje imperativo.

Básicamente la principal funcionalidad de un motor de reglas se reducir de manera simplista a ejecutar un conjunto de reglas basadas en los hechos que definen el estado del problema. Sin embargo si intentamos utilizar motores de reglas en un entorno de trabajo necesitamos una serie de herramientas que faciliten las tareas necesarias para poder

construir el conjunto de reglas. Las funcionalidades que deben cubrir estas herramientas son muy similares a las que puede tener cualquier herramienta que se emplee en el desarrollo de aplicaciones informáticas, esencialmente: un repositorio donde almacenar y gestionar las reglas, un editor que facilite la creación de reglas, un entorno integrado para poder ejecutar y probar las reglas y mecanismos que faciliten la integración con el motor de reglas. A los sistemas que cumplen estos requisitos de manera integrada se les suele denominar sistemas gestores de reglas de negocio, más conocidos por sus siglas en inglés *BRMS* (Business rule management system) [6].

Sobre el repositorio, en cuanto se empieza a trabajar con reglas es imprescindible un lugar donde almacenarlas y gestionar su almacenamiento. Obviamente persistir las reglas es una necesidad muy clara, no obstante las reglas son un elemento imprescindible de la aplicación, por lo que no sólo necesitamos poder guardar las reglas en un lugar seguro y accesible, sino también poder gestionar diferentes versiones de una misma regla (poder volver a una versión anterior en caso de cometer algún error); poder etiquetar conjuntos de reglas (agrupar las reglas por versiones); autentificar qué usuarios pueden almacenar dichas reglas, incluso llevar un control de las operaciones que realizan los usuarios sobre el conjunto de reglas. Todas estas funcionalidades son las que cubren los repositorios.

Un editor de reglas puede parecer una funcionalidad poco importante, puesto que los lenguajes utilizados para definir reglas tienen una sintaxis bastante sencilla. Sin embargo escribir las reglas no suele ser una tarea tan sencilla como sería deseable. El objetivo de un editor de reglas es facilitar la creación y modificación de una regla. Existen herramientas en este sentido de diversa naturaleza, desde restrictivos editores guiados que exigen muy poco conocimiento sobre el lenguaje utilizado para definir reglas, hasta editores abiertos donde se puede escribir cualquier cosa, y se limitan tan sólo a validar la sintaxis escrita, pero que exigen un gran conocimiento del lenguaje. Cuanto más complejo es el lenguaje en el que se escriben las reglas más importancia tiene el editor. Por citar un ejemplo, el BRMS, WebSphere ILOG JRules (http://www-01.ibm.com/software/es/websphere/ilog/), permite escribir algunas reglas utilizando lenguaje natural.

Además de lo expuesto, todo BRMS debe tener incorporado un motor de inferencia que sea capaz de ejecutar el conjunto de reglas que permite generar. El objetivo de este motor, es permitir realizar tests o simulaciones sobre las reglas que se están desarrollando, independientemente del resto del sistema, así el entorno de tests de reglas estaría completamente protegido, además de ser más sencillo realizar pruebas. Bajo determinadas arquitecturas, el BRMS podría ser empleado como el propio motor de reglas del sistema. Incluso un mismo BRMS puede servir de motor a varios sistemas.

También es una funcionalidad imprescindible que incluya mecanismos que permitan de una manera sencilla integrarse con el resto del sistema. Es decir es su responsabilidad tener los mecanismos que puedan facilitar, invocar de manera remota el motor o acceder a la base de reglas para que un motor externo las pueda cargar.

Además de sus funcionalidades básicas, los BRMS, suelen incluir algunas funcional-

idades extras orientadas a mejorar la productividad del trabajo con sistemas de reglas. Por citar algunos ejemplos: incluir herramientas para gestionar ”proyectos”de reglas; gestionar categorías para las reglas; indexar las reglas para mejorar su localización; acreditar el acceso, creando diferentes perfiles de usuario...

Algunos de los motores de reglas más importantes, son:

- **CLIPS:** Fue una de las primeras herramientas para la elaboración de sistemas basados en el conocimiento. Se puede utilizar como motor de reglas.
- **Jess:** Es un motor de reglas escrito enteramente en Java. Es la implementación de referencia del jsr-94 [5].
- **WebSphere ILOG JRules:** Es un BRMS comercial, desarrollado por IBM. Está muy orientado a su implantación en entornos empresariales, así que incluye traducción de sus reglas a diversos formatos, incluyendo lenguajes de programación como COBOL.
- **Drools:** es un conjunto de sistemas que tienen en común que utilizan un motor de reglas, entre las que se incluye un BRMS. Más adelante lo detallaremos en profundidad.

Muchos sistemas basados en el conocimiento utilizan reglas para poder representar estructuras de conocimientos. Como se ha descrito, las reglas son una forma de representar lógica con la que pueden trabajar perfectamente los ordenadores, por ello son bastante populares.

2.3. La plataforma basada en reglas, drools

Drools en su origen era simplemente un motor de reglas, que fue evolucionando hasta que en la versión 4.x incluyó por primera vez las funcionalidades de un BRMS. Actualmente en la versión 5.x ha evolucionado para incluir más funcionalidades, como la gestión de eventos, construir estructuras basadas en reglas... con todas estas funcionalidades se han definido a sí mismos como una plataforma para la *integración de lógica de negocio* (en sus siglas en inglés *BLiP*). Actualmente los módulos principales:

- **Drools expert:** es básicamente el motor de reglas.
- **Drools fusion:** es un procesador de eventos complejos. Utiliza reglas para poder detectar patrones complejos, en tareas tales como: la monitorización de procesos, comunicación entre aplicaciones empresariales..., permite asociar eventos a patrones y lanza los eventos bajo la aparición de diferentes patrones.
- **Drools flow:** combina reglas y tareas juntas. Es decir permite gestionar tareas incluyendo procesos de decisión para ejecutar dichas tareas.
- **Drools guvnor:** cubre las funcionalidades de un BRMS.

- **Drools solver:** es un algoritmo de búsqueda que utiliza todas las herramientas de los demás proyectos para resolver problemas de planificación. En la versión 5.1 le han cambiado el nombre por *drools planner*.

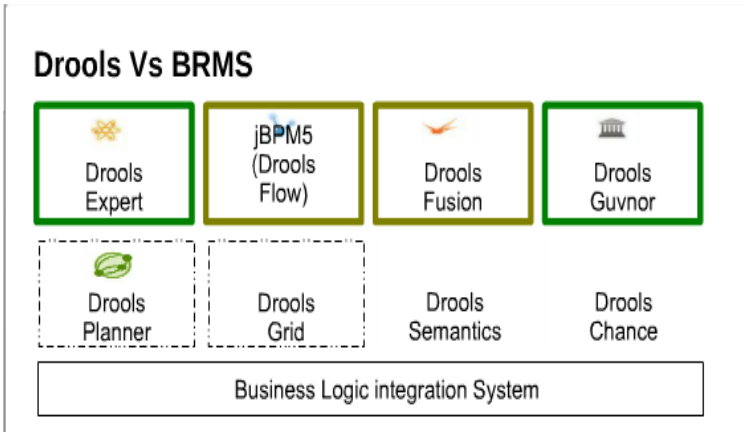


Figura 2.4: Módulos de drools.

Aunque los mencionados son los módulos principales de drools, existen una serie de productos que se encuentran en fases menos maduras que los mencionados. En la figura 2.4 podemos ver todo el ecosistema de productos sobre el que drools se está trabajando. El elemento común en todos los módulos es el motor de reglas. Los cuatro más importantes serían: **drools expert**, **drools flow**, **drools fussion** y **drools guvnor**; menos importantes y en un estado de desarrollo inferior se encontrarían **drools planner** y **drools grid**; por último todavía en fase de conceptualización, estarían los proyectos: **drools semantics** y **drools chance**. Además de los módulos mencionados existen una serie de subproyectos que buscan mejorar la experiencia de trabajo con drools, algunos de éstos son: **drools CLIPS**, que persigue poder traducir reglas escritas para el motor CLIPS en reglas entendibles por el motor drools; **drools plugin**, que consiste en un plugin para el IDE Eclipse, con el fin de proveer un entorno de desarrollo de reglas de drools en dicho IDE; y **drools jsr94**, que es una implementación del jsr94, de tal manera que implementa la interfaz estándar definida por el jsr, con la funcionalidad que ya ofrece drools.

Todos los módulos de drools están desarrollados en Java utilizando tecnologías JEE (Java Enterprise Edition).

El motor de reglas, drools-expert

Dentro del motor de reglas, debemos saber que existen diversas formas de poder transcribir las reglas: en XML o en lenguajes de definición de dominio propios de drools (drl y dsl). En las primeras versiones de drools sólo se podían escribir las reglas en *XML* (Extensible Markup Language), pero era bastante complicado y dificultaba la evolución del lenguaje de reglas por incompatibilidades entre diferentes versiones. Como evolución se introdujo *drl*. Drl es un lenguaje muy completo con una potencia enorme, sin embargo su

sintaxis es mucho más cercana a la sintaxis de un lenguaje de programación imperativa, que a los conocimientos que suelen tener las personas que editan reglas, así que se ideó *dsl* como una forma de definir reglas para que fuesen editadas de manera sencilla. Aunque se permite escribir reglas de diversas formas, al final se traducen las reglas a una estructura que es idéntica independientemente de cómo se escribiesen las reglas. De esta forma existe un único motor y diferentes métodos para escribir y validar las reglas. Podemos observar rápidamente las diferencias entre estas tres formas de representar reglas en la figura 2.5.

Regla escrita en xml:

```
<package name="com.sample"
         xmlns="http://drools.org/drools-4.0"
         xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
         xs:schemaLocation="http://drools.org/drools-4.0 drools-
4.0.xsd">
      <import name="org.drools.*" />
      <rule name="Hello_world">
            <lhs>
                  <pattern identifier="m" object-type="Message" >
                        <field-constraint field-name="status">
                              <literal-restriction evaluator="=="
identifier="Message.HELLO" />
                                    <field-binding field-name="message"
identifier="message" />
                        </field-constraint>
                  </pattern>
            <rhs>
                  System.out.println( message );
                  modify ( m ) { message = "Goodbyte cruel world",
                           status = Message.GOODBYE };
            </rhs>
      </rule>
</package>
```

Regla escrita en drl:

```
rule "Hello World"
      dialect "mvel"
  when
      m : Message( status == Message.HELLO, message : message )
  then
      System.out.println( message );
      modify ( m ) \{ message = "Goodbyte cruel world",
                      status = Message.GOODBYE \};
end
```

Regla escrita en dsl:

```
[when]There is a Message with {status} =
Message(Message=="{Message.HELLO}", "{message}"=message)
[then]Log "{message}"=System.out.println("{message}");
```

Figura 2.5: Hola Mundo escrito en los diferentes formatos de reglas de drools

Para definir el modelo se pueden utilizar objetos programados en Java. Ésto facilita la integración del motor de reglas dentro de una plataforma jee, ya que puede utilizar los objetos de dominio como ontología del problema. Aunque no es necesario ya que también se pueden definir utilizando una sintaxis especificada por drools. Debemos sumarle

Modelo escrito en Java:

```
public static class Message {
    public static final int HELLO   = 0;
    public static final int GOODBYE = 1;

    private String          message;
    private int             status;
}
```

Modelo escrito en drl:

```
declare Message
      message : String
      status: int
end
global int HELLO = 0;
global int GOODBYE = 1;
```

Figura 2.6: Modelo escrito para el motor de drools en Java y en drl

a ésto, que los tipos que maneja la ontología son los mismos que Java. En este caso las recomendaciones sobre tipado en drools es seguir las recomendaciones emitidas por los desarrolladores de Java. Por supuesto esto implica que los hechos dentro de drools tienen todas las características de los objetos Java, herencia, polimorfismo y abstracción. Una clase Java está formada por su nombre, sus atributos, y sus métodos. Los métodos de una clase de dominio pueden ser invocados dentro de las reglas de drools. La sintaxis de drools para definir el modelo no tiene soporte para funciones. El modelo de drools es de tipado estricto. En la figura 2.6 podemos ver una entidad de un modelo representada en Java y en el lenguaje de definición de drools.

Un aspecto muy importante a tener en cuenta sobre la integración del motor de reglas en una aplicación es si hemos construido el modelo utilizando drools, es que a la hora de introducir los hechos no podrán ser objetos del modelo de datos que emplea la aplicación (como podría ser el caso si el modelo de drools fuese importado de clases Java), es decir que será necesario algún tipo de procedimiento que nos permita transformar los objetos que maneja la aplicación en objetos que maneja el motor de reglas. Drools ya incluye algunos mecanismos para facilitar esta tarea.

La frontera entre Java y drools es en ocasiones bastante difusa, permitiendo por ejemplo importar clases programadas en Java dentro de un paquete de reglas, invocar los métodos de dichas clases o utilizar constantes definidas. Incluso podremos utilizar sentencias Java dentro de nuestras reglas. Además podremos emplear como hechos objetos creados en Java, ya que el motor representa internamente los hechos de esta forma.

Escribir reglas en drools utilizando XML es bastante complejo. El principal problema es que hay que utilizar siempre una DTD (Document Type Definition) lo que puede originarnos problemas de compatibilidad entre diferentes versiones de drools. Es imprescindible tener un conocimiento a fondo de la DTD. Existen unas diferencias mínimas (al nivel de operaciones a realizar) entre utilizar la sintaxis de XML en vez de la definida con drl. Sin embargo la sintaxis con drl es mucho más sencilla.

Tanto en la sintaxis basada en XML como en drl, se pueden definir funciones, constantes y la ontología del problema.

La sintaxis de las reglas escritas en drl es bastante sencilla, la definición de patrones es muy intuitiva. Gracias a la librería MVEL (http://mvel.codehaus.org/) se pueden utilizar expresiones basadas en EL (expression language), que se utilizan habitualmente por programadores de JAVA en la construcción de ficheros .jsp, entre otras situaciones. EL está considerado como una manera bastante intuitiva y sencilla de construir sentencias de programación. La utilización de estos elementos permite que un desarrollador JEE pueda familiarizarse rápidamente con las reglas de drools escritas en drl.

Algunas de las principales características del lenguaje drl para definir reglas, además de las ya mencionadas, son:

- concatenar fácilmente restricciones basadas en campos.
- acceder a información alojada en mapas.
- utilizar expresiones para evaluar restricciones.
- almacenar restricciones en variables.
- comprobar restricciones sobre todos los hechos.
- comprobar restricciones sobre colecciones de objetos.
- realizar operaciones sobre colecciones de objetos.
- operaciones lógicas y matemáticas.

Destacar sobre todo la importancia de los operadores *from* y *forall* que se utilizan para trabajar sobre colecciones de elementos. From realiza una operación sobre el conjunto de elementos de una colección, y permite que esa colección se filtre por un patrón concreto. Es especialmente útil para poder realizar operaciones de agregados sobre conjuntos de elementos, tales como calcular sumatorios, promedios, recuentos... No es un operador lógico, aunque se puede incluir evaluaciones booleanas. Dada la importancia de los sumatorios y otras operaciones de agregación que se incluyen dentro del simulador de SIMBA, éste operador es muy importante para solventarlo de una manera sencilla y eficaz. El operador forall nos permite realizar una comprobación booleana sobre una colección de elementos, siendo el resultado verdadero (si todos los elementos cumple con la condición) o falso. Los dos operadores permiten crear una colección a partir de la base de hechos, con los que verifiquen cierto patrón, permitiendo de esta forma que la colección no sea un hecho en sí misma.

Otro aspecto interesante de drools-expert es que permite modificar/crear hechos, sin que éstos cambios afecte a la memoria de trabajo y por tanto provoquen que determinadas reglas se disparen.

Una parte muy importante de las reglas es la meta-información y los atributos. La meta-información es información que no es procesada por el motor, pero que ayuda a las personas a comprender mejor la regla. Los atributos nos describen condiciones que no son propias del dominio pero que son igualmente importantes a la hora de activar una regla. Algunos de los más importantes son:

- **no-loop:** no permite activar una regla más de una vez.

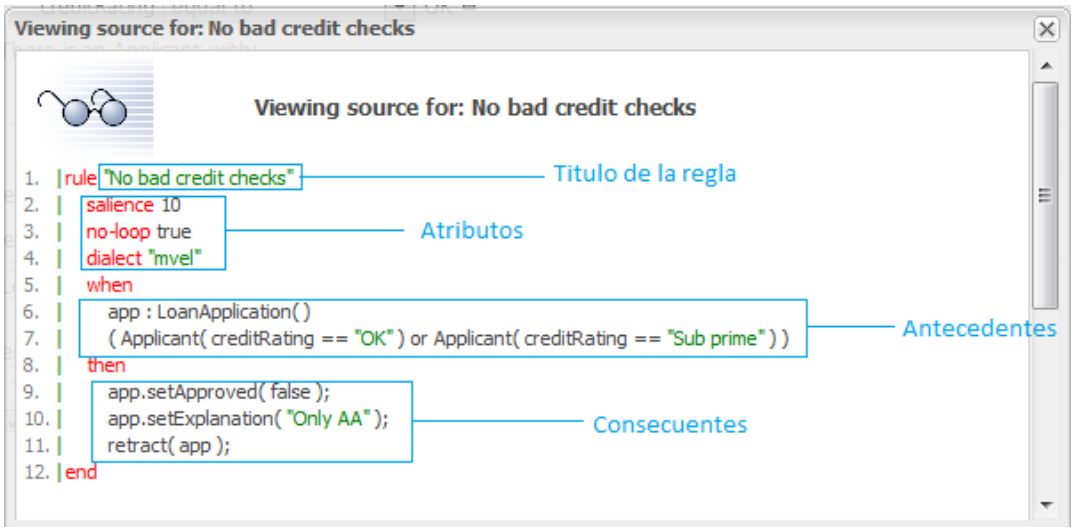


Figura 2.7: Principales elementos de una regla con la sintaxis drl.

- **salience:** prioridad de una regla.
- **activation-group:** identifica un grupo de activación, de tal forma que de todas las reglas que pertenezcan a un mismo grupo sólo se puede activar una.
- **date-effective:** marca la fecha a partir de la cual una regla puede ser activada.
- **date-expires:** marca la fecha a partir de la cual una regla no puede ser activada.

La figura 2.1 representa una regla escrita en drl. Es una regla sobre el dominio de concesión de créditos bancarios y trata de determinar si las garantías de la entidad que va a recibir el crédito son suficientemente buenos o no. Lo importante de la figura es resaltar los principales elementos que forman parte de una regla, titulo, atributos (en este caso salience, no-loop y dialect), antecedentes (en este caso tenemos dos patrones, uno para cargar los hechos del tipo *LoanApplication* y otro para evaluar dos atributos del hecho) y consecuentes (que este caso son modificaciones de los hechos que se cargan en los antecedentes).

En cuanto a la otra forma de definir reglas, dsl, no es una sintaxis exactamente, es una definición de reglas que podemos hacer para permitir que personas que no conocen drl puedan crear reglas. Por tanto la definición de un conjunto de reglas programadas en dsl tiene dos pasos, por un lado hay que crear los patrones que cubran los tipos de reglas que quieren hacerse y por otro escribir las reglas que cumplan esos patrones. Es muy útil, cuando reglas que queremos aplicar siempre responden a un patrón sencillo (una plantilla) y dicho patrón cubrirá ampliamente el espectro de reglas que se van a usar; cuando queremos ajustar el lenguaje de reglas a la gente que trabaja con el dominio, o si por seguridad se desea limitar las posibilidades de creación de reglas con drools. La principal utilidad es que los patrones pueden conseguir que se escriban reglas en lenguaje natural. La explicación de las resultados queda muy detallada al poderse leer la secuencia ejecución de las reglas en lenguaje natural.

No se pueden crear plantillas para drools dsl que permitan elementos tan completos como los que se pueden escribir encontrar en las reglas definidas con drl.

El motor de inferencia de drools implementa el algoritmo Rete, del que ya hemos hablando anteriormente. Destacar que en caso de colisión de varias reglas, la regla a ejecutar se decide mediante la utilización del atributo *salience*, de forma que la regla con mayor valor en éste atributo será la escogida.

Respecto a la integración de drools-expert con otros sistemas, hay que tener en cuenta que tiene una implementación de la especificación jsr94, además se incluye una completa API (en inglés, application programming interface) para poder invocar y configurar el motor en un programa Java.

El BRMS, drools-guvnor

Drools-guvnor es el sistema que se emplea para cubrir las necesidades de un BRMS dentro del ecosistema de drools. Uno de los principales objetivos es valorar el uso de la herramienta drools-guvnor como BRMS, a la hora de construir un conjunto de reglas. Las funcionalidades más destacados que incluye son:

- **Gestión de proyectos:** permite crear proyectos de reglas, compilarlos y etiquetarlos.
- **Edición de reglas:** incluye diversos procedimientos para poder crear reglas.
- **Gestión de usuarios:** trae mecanismos para validar usuarios.
- **Control de versiones:** almacena todas las modificaciones salvadas de un regla (y de un proyecto), permitiendo poder gestionar diferentes versiones (sustituir una regla por una versión anterior, comprobar el log de cambios...).
- **Gestión de meta-información:** Para las reglas maneja más meta-información de la requerida por el motor, no tan importante para sus despliegue como para su gestión (en qué fecha se realizó, quién fue, en qué categoría la clasificó...).

Para poder crear reglas, el primer paso es crear un proyecto y una ontología. Para crearla existen dos caminos, por un lado importar un fichero de extensión .jar, que contenga las clases Java que definen en la ontología; por otro, podemos utilizar un editor guiado que nos permita crear las estructuras que formen parte del modelo. Lo importante de la edición guiada es que exige muy pocos conocimientos sobre drools. Para la elaboración de la ontología, es necesario definir mediante un formulario una entidad y los atributos (con su correspondiente tipo) de dicha entidad. Los tipos de los atributos se definen en Java por lo que es necesario conocer las peculiaridades de los tipos que permite este lenguaje, aunque se ofrecen algunas de los tipos más habituales como sugerencia dentro del editor guiado. Después de crear de la ontología, más adelante podremos utilizar como tipo de dato éste elemento, siguiendo la filosofía de la programación orientada a objetos. El resultado del

editor guiado no es un fichero jar, sino una definición de objetos en drl. El editor guiado no permite establecer relaciones de herencia entre entidades. Otro elemento reseñable, es la posible utilización de enumerados como tipo de datos. Estos enumerados no afectan en ningún momento a la ontología creada, pero durante la elaboración de reglas, limitarán y facilitarán la selección de posibles valores dentro de un patrón.

Una vez que hemos creado el modelo, podremos crear reglas que utilicen restricciones sobre el modelo que hemos definido. Las principales formas de crear reglas son: *editor guiado*, editor libre de código, un flujo de reglas (drools-flow), matrices de decisión y matrices de decisión guiadas.

En el caso de que se decida prescindir de la posibilidad que ofrece drools-guvnor de editar de manera guiada las reglas, o que ya se tenga un importante número de reglas escritas y se desee incorporarlas al aplicativo, se nos ofrece la posibilidad de importar reglas ya escritas y desarrollar nuevas, directamente en los lenguajes de definición que usa drools (drl y dsl), salvo XML. Los flujos de reglas se deben importar a partir del editor de flujos de reglas que se crean en Eclipse con un plugin de drools (el ya mencionado drools-plugin).

Dentro de drools-guvnor existen editores libres, que permiten escribir reglas en cualquier formato. Sin embargo estos editores no incluyen ningún tipo de ayuda en lo referente al modelo que se ha creado, restricciones en las comparaciones según el tipo de dato, etc ....

Las matrices de decisión son estructuras, con una matriz que cruza atributos en un eje y consecuentes en el otro, de tal manera que nos permite decir un valor para cada atributo con el que se activa el consecuente. De esta forma podemos definir reglas sencillas, pero en muy poco tiempo podemos tener un conjunto enorme de reglas. Esta estructura es muy útil para resolver determinados problemas.

La manera de crear reglas más interesante, es la utilización del editor guiado. El editor guiado consiste en rellenar la información de la regla mediante un formulario que de como resultado una regla sin errores sintácticos. Realmente no es del todo cierto, ya que el margen que permite el editor sí que puede dar pie a cometer ciertos errores sintácticos. Éste será el editor que se evaluará en el estudio, puesto que en un entorno real los usuarios que crean las reglas no deberían tener la obligación de conocer completamente la sintaxis con la que se tienen que escribir.

Podremos modificar cualquier regla en el mismo editor donde se creó. Además podemos copiarlas, modificarlas, modificar metainformación (salvo la referente a la creación), verificar la sintaxis generada o ver el código generado (en drl).

En drools-guvnor existen herramientas para crear funciones (no guiadas), procesos (de nuevo como apoyo a drools flow), entre otros elementos del entorno de drools.

Junto con la creación de reglas, podemos crear casos de test para las mismas y de estas maneras podemos comprobar que se comportan como han sido diseñadas. Para crear el caso de test, simplemente hay que definir el estado de entrada, el estado final, el conjunto de reglas que queremos probar y un conjunto de opciones para limitar el caso de test (qué reglas se pueden seleccionar, simular una fecha...).

Junto con las reglas es muy interesante la meta-información que se guarda de cada regla. En especial destacar las categorías, que se consideran un elemento de administración. Las categorías nos permiten agrupar reglas para poder localizarlas más rápidamente, pero no tienen ningún efecto sobre el motor. Podemos llegar a utilizar las mismas categorías en diferentes proyectos de reglas. Destacar que incluso una regla puede tener varias categorías.

Otra meta-información muy útil es el estado de la regla. Podemos crear estados para dar información a los otros usuarios del aplicativo sobre la regla. Un problema real muy importante en los BRMS es la carencia de mecanismos para comunicarse dentro de la herramienta el trabajo que se está realizando, para solventar este problema se pueden utilizar los estados.

En cualquier entorno de desarrollo es crítico el uso de un gestor de versiones. Las funcionalidades que ofrece guvnor como gestor de versiones son: el almacenamiento de cada regla y cada cambio que se haya salvado; añadir un información de cada cambio; poder recuperar alguna versión anterior de una regla; y agrupar todas las reglas que han sido editadas en una versión lista para desplegar; etiquetar un conjunto de reglas; recuperar otras etiquetas. Un problema bastante común cuando se trabaja con reglas es que todas las reglas se encuentra en un único fichero (o en unos pocos), por lo que utilizar un gestor de versiones que se emplea en el desarrollo del software no suele ser una buena opción, ya que al estar todas las reglas en el mismo fichero se presentan dos problemas fundamentalmente, por un lado puede que necesitemos volver una regla varias versiones hacia atrás, al estar todas en el mismo fichero la operación se puede complicar bastante; otro problema importante surge del trabajo en equipo, si varias personas están modificando diferentes reglas la tarea de unificar el fichero puede ser bastante compleja. Utilizar un fichero para salvar cada regla tampoco suele ser una buena idea puesto que un sistema basado en reglas puede llegar a tener un tamaño tan grande de reglas que manejarlo de ésta forma puede ser imposible.

Drools-guvnor utiliza un repositorio basado en la definición estándar de Java jsr-170, lo que permite poder cambiar el soporte físico sobre el que queremos que se almacene la información vía configuración. Por defecto el soporte es sobre disco, pero no es completo, por ejemplo no almacena todas las versiones de una regla. La configuración no se hace dentro de la aplicación, sino que es necesario realizar los cambios en los ficheros de configuración de la aplicación.

En un BRMS la gestión de usuarios es muy importante. Puesto que existen tareas que se pueden llevar a cabo y tienen consecuencias imprevisibles (por ejemplo la eliminación de un elemento de la ontología), es importante comprobar que sólo pueda entrar en la aplicación quien tenga privilegios para ello. Además dentro de la aplicación hay que garantizar que

los usuarios sólo realicen las tareas para las que poseen conocimientos. Vía configuración, drools-guvnor permite incluir mecanismos de validación de usuarios (validación con LDAP, Lightweight Directory Access Protocol), aunque en ocasiones implicará también un pequeño desarrollo (validación contra base de datos que no se ajusta a los estándares). Para la asignación de los permisos se realizará vía administración. Los tipos de usuario que se pueden configurar son: administrador, analista, analista de sólo lectura, o asociados a un proyecto de reglas. Por defecto (si no se ha configurado ningún método de validación) cualquier usuario que trate de hacer login lo hará con permisos de administrador. Al igual que en el repositorio, cambiar el método de validación exige modificar los ficheros de configuración.

En cuanto a la implementación de drools-guvnor, es una aplicación JEE, que está desarrollada utilizando GWT (Google Web Toolkit) para desarrollar un cliente RIA (rich internet applications) en javascript que se comunica utilizando el protocolo RPC (Remote Procedure Call) con la parte servidora. El servidor está desarrollado en JEE, utilizando el framework Jboss Seam.

Referente a la arquitectura (figura 2.6), la parte de presentación de la aplicación es la que muestra las opciones de desarrollo y gestión de reglas, se apoya en un motor de inferencia (drools-expert, se emplea para realizar labores de compilación, pruebas...) y en un gestor de eventos complejo (drools-fussion, es importante para establecer mecanismo de comunicación, gestión de versiones...). Toda la información que tiene que ser persistida se almacena en un repositorio. Finalmente necesita un servidor de aplicaciones JEE para ejecutarse.

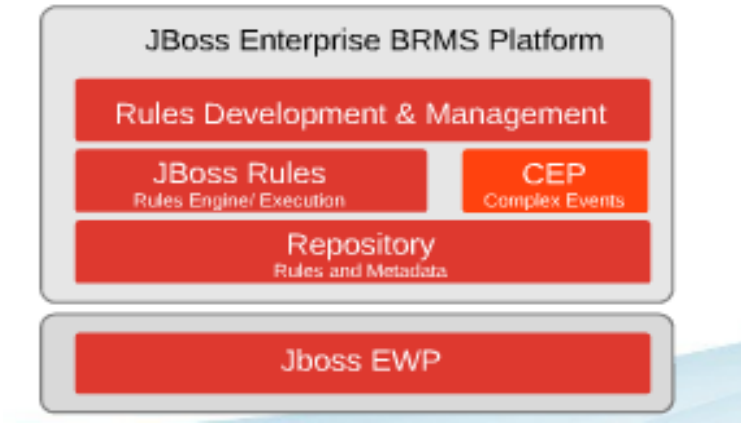


Figura 2.8: Arquitectura de drools-guvnor.

2.4. CLIPS, una solución para construir sistemas basados en el conocimiento

CLIPS (C Language Integrated Production System) es una herramienta para construir sistemas expertos, probablemente una de las primeras (sus orígenes datan de 1984). Su desarrollo fue iniciado por el departamento de inteligencia artificial de la NASA. Cuando empezaron a construir la herramienta intentaron emplear Lisp, sin embargo se encontraron con grandes inconvenientes en su desarrollo (alto coste en las herramientas, dificultad para integrarlo en otros sistemas...). Por ello enfocaron el desarrollo basándose en otro lenguaje de programación, en concreto en C. Tras iniciar el desarrollo, la primera versión de CLIPS vio la luz en 1986.

Algunas de las principales características de CLIPS son:

- **Representación del conocimiento**: provee de una herramienta que soporta diferentes formas para representar el conocimiento, en concreto: basada en reglas, orientado a objetos o procedimental.
- **Portabilidad**: por razones de portabilidad y eficiencia CLIPS está escrito en C y puede ser instalado en diferentes sistemas operativos sin necesidad de modificar el código fuente. En particular puede utilizarse en cualquier sistema operativo que donde exista un compilador de ANSI C.
- **Integración/Extensivo**: puede ser embebido en cualquier sistema, invocándolo como una subrutina. En concreto en aplicaciones desarrolladas en C sería muy sencillo, pero también se incluye soporte para los lenguajes de programación: Java, FORTRAN o ADA.
- **Desarrollo interactivo**: la versión estándar de CLIPS ofrece un entorno de desarrollo que ofrece ayuda, debug y un editor.
- **Bajo coste**: es software de dominio público.

Como se puede ver CLIPS utiliza reglas para representar el conocimiento, además de otros elementos para definir hechos. Para crear una ontología que soporte el problema que se intenta resolver, podemos utilizar estructuras con propiedades similares a las de la programación orientada a objetos, como son la abstracción, la herencia y el polimorfismo. Estas herramientas se denominan *marcos* (utilizan la palabra reservada deffclass para definirse). También permite otras estructuras denominadas *plantillas* (se definen con deftemplates), que aunque muy similares no tienen las propiedades de herencia, abstracción y polimorfismo (figura 2.9). Utilicemos el elemento que utilicemos debemos ser conscientes de que tiene que tener un nombre único y una serie de atributos. Para cada atributo, podemos definir un valor por defecto y un tipo, aunque ninguno es obligatorio.

El lenguaje específico de dominio que define CLIPS para escribir las reglas tiene una sintaxis muy similar al lenguaje de programación Lisp, que está basado en el paradigma de programación funcional. De la misma forma que drools en el antecedente permite utilizar diversos patrones, evaluaciones, y funciones. Igualmente en el consecuente se opera sobre la

```
(deftemplate object
  (slot name (default ?NONE))
  (slot location)
  (slot on-top-of)
  (single-slot weight
    (type FLOAT))
  (multislot contents))
```

Figura 2.9: Ejemplo de ontología escrito en CLIPS. En este ejemplo se ven cómo se define un objeto en el lenguaje de definición de onotología de CLIPS. Algunas de sus principales caracteristicas: definición de atributos (*name*, *location*, *on-top-of*, *wieght* y *contents*), permitir especificar el tipo del atributo (*type FLOAT*), un valor por defecto (*default ?NONE*) o si es un atributo con un único valor (*slot* o *single-slot*) o múltiples (*multislot*).

base de hechos o se invoca funciones. Se puede ver un ejemplo de regla escrita para CLIPS en la figura 2.10.

```
(defrule example-rule "This is an example of a simple rule"
  (refrigerator light on)
  (refrigerator door open)
  =>
  (assert (refrigerator food spoiled)))
```

Figura 2.10: Ejemplo de regla escrita en CLIPS. En este ejemplo sencillo podemos observar las principales parte de una regla, el título, los patrones que forma parte del antecendete (por encima del símbolo =>) y el consecuente que se activa cuando los patrones son ciertos (por debajo de =>).

De manera similar a como funciona drools, para decidir cuáles son las reglas que se pueden activar en cada momento el motor de inferencia implementa el algoritmo Rete. Igualmente, la estrategia de resolución de conflictos se basa en un atributo numérico (denominado *salience*), de manera que la regla que tenga mayor valor en este atributo es la regla que se selecciona.

Ya se realizó un trabajo previo similar evaluando como motor de reglas CLIPS en vez de drools. Sin embargo en CLIPS no existen herramientas que nos permitan cubrir las funcionalidades que ofrece un BRMS de manera sencilla. Además de ésta importante diferencia con el entorno de drools existen otras. Por ejemplo, drools permite una integración muy sencilla a través de una API en Java, en el caso de CLIPS esta integración hay que hacerla a través de una API en C (por ejemplo la integración con Java se lleva a cabo con JNI, que es básicamente un wrapper sobre una librería escrita en otro lenguaje). Mientras la sintaxis para escribir patrones en el antecedente y los consecuentes de la regla de CLIPS es compleja (muy similar al lenguaje de programación Lisp), en drools la sintaxis es mucho

más sencilla basada en EL y en otros elementos similares a Java.

En la sintaxis de drools existen algunos elementos que facilitan ciertas operaciones como por ejemplo los operadores from y forall. Además drools exige un tipado bastante estricto, mientras que CLIPS no lo considera obligatorio.

2.5. Java Enterprise Edition

Puesto que drools está desarrollado en JEE y todo el desarrollo necesario se va llevar a cabo usando esta tecnología, parece apropiado hacer una pequeña reseña.

Java Enterprise Edition (más conocido como JEE) es una plataforma de programación (parte de la plataforma Java) para desarrollar y ejecutar software de aplicaciones en lenguaje de programación Java con arquitectura de N capas distribuidas y que se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. La plataforma JEE está definida por una especificación, que define cómo deben de ser las aplicaciones desarrolladas en JEE, y qué servicios (y cómo) deben ofrecer los servidores de aplicaciones que quieran ejecutar este tipo de aplicaciones. Comúnmente se dice que las especificaciones forman el estándar JEE ([13]).

Además de definir cómo deben de ser los servidores de aplicaciones y las aplicaciones, también son parte de JEE aquellas APIs que se se consideran imprescindibles (conexión a base de datos...) o aquellas que por su uso están muy extendidas (repositorios, orms, inyección de dependencias, brms...). Para decidir cómo deben de ser dichas librerías se reune a expertos en el tema que trata de cubrir la librería, junto con la comunidad, lo que se conoce como java community process. El resultado de todo ésto se plasma en lo que se conoce como un jsr, *Java Specification Requests*, que es un documento formal que detalla las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma Java, una implementación de referencia (que es una implementación de código libre de la necesidad que se intenta cubrir) y una API para garantizar la independencia de la implementación concreta que se emplee. Como hemos comentado el jsr que define la interfaz de un motor de reglas y de la que drools tiene una implementación es el jsr-94 ([5]).

El servidor de aplicaciones en el que se va a desplegar drools-guvnor jBoss Application Server. Aunque en principio se podría desplegar en cualquier otro servidor de aplicaciones que cumpla con las especificaciones JEE.

2.6. Simuladores empresariales

Los simuladores permiten estudiar comportamientos en entornos donde es muy costoso medir la situación real, ya sea por el riesgo que conlleva una medición en un entorno real o por la necesidad de comprobar como afecta nuestro comportamiento al entorno de una

manera segura. Se suelen emplear parar representar entornos de naturaleza caótica.

Los sistemas basados en conocimientos que se emplean en entornos financieros suelen incluir simulaciones, ya que muchos problemas que tratan de resolver están relacionados con vistas a futuros y además son sistemas caóticos.

Por resaltar el contexto de uso de SIMBA, el uso de simuladores empresariales para educación, pretende facilitar el paso de los conocimientos que poseen los alumnos a utilizar esos conocimientos en un entorno de decisión lo suficientemente parecido a un entorno real, sin tener los riesgos asociados. De esta forma se consigue una mejor formación y una reducción de los costes de los mismos. Los resultados indican que los estudiantes que utilizan estos sistemas mejoran en las capacidades de gestión de organizaciones, creatividad, criterio, interacción y discusión.

La simulación de la gestión empresarial es un problema cuya solución es muy compleja puesto que existen múltiples áreas funcionales cuyo funcionamiento se ve interrelacionado. Sin embargo los simuladores se suelen centrar principalmente en un área (generalmente área comercial), limitando los resultados del mismo. También el número de variables que se puede llegar a utilizar puede ser enorme, sin embargo se limitan siempre a las que se consideran más significativas, pero de igual manera ésto puede limitar el resultado de la simulación.

Algunos simuladores intentan un enfoque de aproximación dinámica al problema (como es el caso de SIMBA), es decir, el estado del problema es la consecuencia de la interacción entre sus elementos. Al hacer esta aproximación, las soluciones que aportan no tienen ni la consistencia ni la persistencia que las soluciones que se extraen de sistemas que utilizan algoritmos complejos. Éstos simuladores, por tanto, no calculan una solución óptima, pero es que en determinados campos (como algunos problemas que presenta la economía) no existen soluciones óptimas. Otro tipo de enfoque de simulación puede ser la construcción de un modelo matemático cerrado que cubra toda la solución del problema.

2.7. El simulador empresarial SIMBA

SIMBA (SIMulation in Business Administration) es un simulador empresarial orientado a ser utilizado en entornos docentes. El principal objetivo es emular el contexto real de los negocios utilizando las mismas variables, relaciones y eventos; y con este contexto permitir que varias equipos (cada uno representa una empresa) puedan competir entre sí.

Básicamente el funcionamiento de SIMBA consiste en que varios equipos, revisan individualmente información sobre el sistema del mercado. En base a esa información toman una serie de decisiones. El simulador evalúa el impacto que tienen las decisiones de cada uno de los equipos sobre el mercado y sobre la respectiva empresa de cada equipo. Este procedimiento modifica el estado del sistema y se realiza un nuevo ciclo de toma de decisiones en función del nuevo estado. Además permite establecer un ranking para poder evaluar a los equipos participantes. Los equipos no toman las decisiones en tiempo real,

sino que toman un conjunto de decisiones y cuando se cumple un periodo de tiempo es cuando se valoran el efecto de las decisiones sobre el entorno.

Algunas de sus principales características es que permite definir múltiples productos, mercados, segmentación de clientes... Desde el punto de vista docente, permite ajustar el nivel de interacción con el sistema a los conocimientos que tengan los usuarios, que forman parte de los equipos, además los usuarios acceden vía web al sistema. Permite localizar la perspectiva de los usuarios...

Desde el punto de vista funcional (figura 2.11), los principales módulos son: *usuarios* (USERS), *personalización* (CUSTOMIZATION), *planificación* (PLANIFICATION), *conocimiento* (KNOWLEDGE) y *simulación* (SIMULATION PROCCESS).

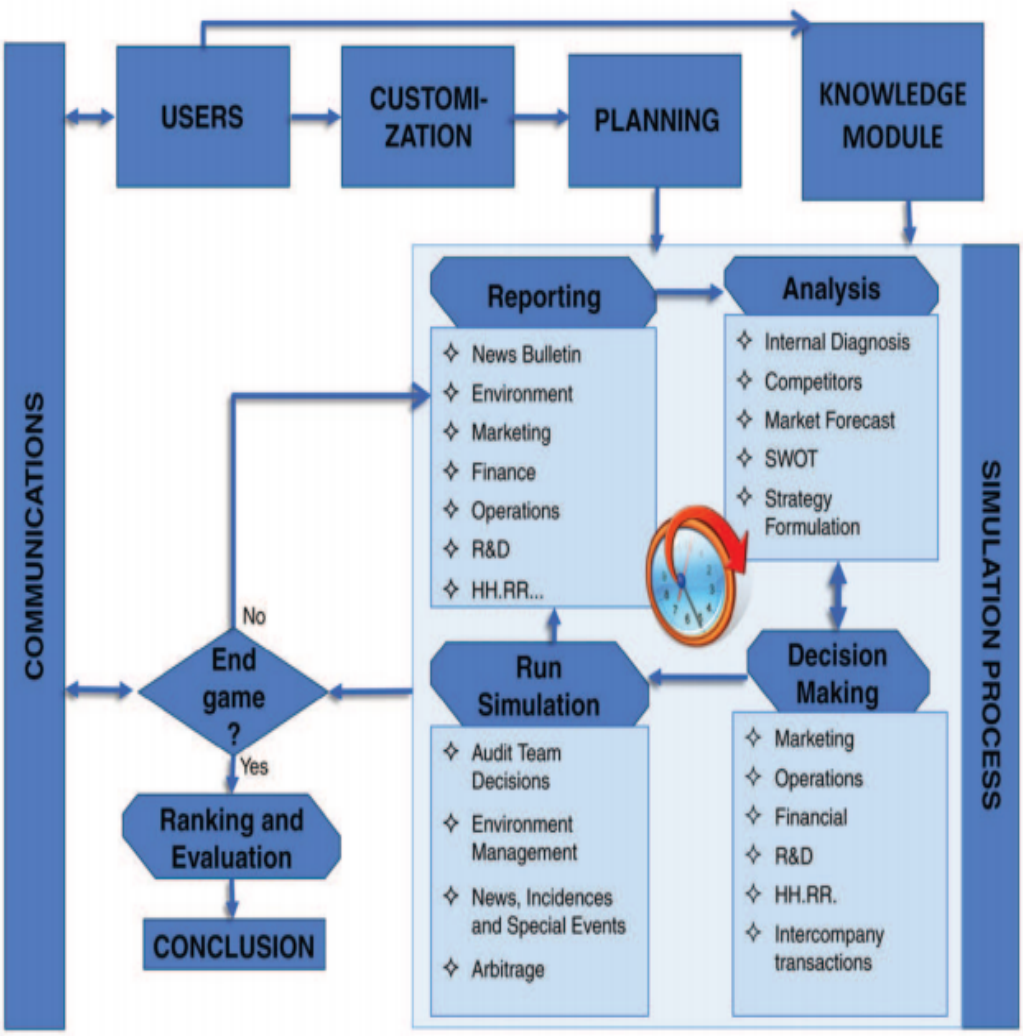


Figura 2.11: Modelo de SIMBA.

El módulo de usuarios permite crear los perfiles de usuarios requeridos por los participantes de SIMBA, a saber clientes, asistentes, instructores, jugadores o administradores. Cada perfil tiene sus propias funciones y responsabilidades dentro de SIMBA.

El módulo de personalización permite modificar aquellas partes de SIMBA que se consideran como elementos configurables, tales como moneda, elementos geográficos, situación inicial, dificultad... Este elemento es clave para poder emplear el simulador en múltiples entornos.

El instructor podrá personalizar el entorno de simulación a través del módulo de planificación. Para ello creará y definirá los mercados, las empresas y los equipos. El módulo de conocimiento provee de soporte a los participantes mientras usan el sistema.

Una vez los equipos están compitiendo, la mayor parte del tiempo están interactuando en el proceso de simulación. Este proceso es por ello el elemento más importante de la arquitectura y puesto que intentaremos sustituir parte del mismo por un motor de reglas basado en drools, ampliaremos su funcionalidad más adelante.

El funcionamiento normal de SIMBA, para los participantes, consistiría en que analizarían la situación, que en base a cómo evalúan la situación toman una serie de decisiones. Cuando todos los equipos han tomado las decisiones sobre un ciclo, se realizan las simulaciones sobre el mercado. Para realizar las simulaciones, los instructores pueden definir condiciones especiales. Una vez se ha realizado la simulación, se elabora una serie de informes sobre el resultado general de la simulación y se da comienzo un nuevo ciclo de decisión. Cuando la competición termina (a criterio de los administradores) se elabora un ranking que ordena el comportamiento de los equipos en el ”juego”. Para elaborar el ranking se utilizan una serie de indicadores que se ponderan según el criterio de los instructores.

Esta es una descripción muy general sobre el funcionamiento de SIMBA, el punto de acción del estudio es el subsistema de simulación dentro del módulo de simulación. Actualmente se utiliza un lenguaje imperativo (c++) para realizar los procesos de simulación. Es aquí dónde se va a intentar evaluar las mejoras que nos propiciaría un motor de reglas, en concreto drools.

Proceso de simulación

Este proceso se encarga de ejecutar la parte más importante de SIMBA. Está compuesto por cuatro módulos específicos y bien diferenciados. Las principales responsabilidades de estos módulos son: proveer la información del estado actual del sistema para que los participantes en el juego puedan tomar sus decisiones. Además registra las decisiones que toman los usuarios sobre sus empresas y ejecuta la simulación que nos llevará al siguiente estado del juego. Por último genera informes para poder analizar el resultados de las decisiones que se han tomado. En función de éstas funcionalidades se divide en cuatro subsistemas: **análisis**, **toma de decisiones**, **simulador** y **generación de informes**. De todos

los subsistemas el que realmente interesa sustituir es el simulador, ya que éste, es el que incluye gran parte de la lógica de la aplicación.

El proceso de simulación empieza con el módulo de generación de informes, a través del cual los equipos obtienen toda la formación de su empresa, sus competidores y las necesidades del entorno. En los módulos de análisis y toma de decisiones, los equipos introducen en el sistema a través de una interfaz de usuario las inversiones que debe realizar su empresa.

El último módulo del proceso es el que realiza la simulación propiamente dicha (módulo de simulación). Éste es el punto central de la competición, se comprueba que todos los equipos han tomado las decisiones oportunas, cuando el periodo para tomar decisiones ha terminado y realiza los cálculos necesarios, cruzando la información de los periodos anteriores con las decisiones tomadas. Una vez realizada ésta tarea se prepara la información para elaborar los informes y que los equipos puedan empezar a trabajar para el siguiente de ciclo de simulación. Al poder cuantificar mediante fórmulas las decisiones de los equipos, los instructores pueden descubrir los errores de los equipos, los hitos que van completando...

A través del módulo de personalización los instructores pueden configurar varios mercados con los que tengan que lidiar los equipos. Para cada mercado definido el módulo de simulación realiza una serie de pasos que son dependientes, éstos son:

- Evaluación del entorno económico. Se evalúan las variables del mercado del contexto económico, tales como la inflación, los ratios de interés, los índices de consumo.
- Evaluación de la demanda del mercado. Se calcula cuál es la capacidad de absorción del mercado que se está trabajando.
- Evaluación de la calidad de la empresas. Esta fase es un primer paso para determinar en que segmento compite cada compañía. Este proceso es bastante más complejo que los demás, debido al gran número de variables que lo afectan. Por ello ha sido seleccionado para el estudio, ya que si se puede resolver correctamente los demás también deberían poder ser resueltos.

Una vez completados estos tres pasos, para cada compañía que participa en el concurso se llevan a cabo los siguiente pasos:

- Cálculo de la demanda potencial. Al comparar las decisiones de marketing de cada equipo con las de las demás del mismo sector en este mercado se calcula cuál puede ser la demanda potencial de cada empresa.
- Cálculo de ventas. Se realiza una comparación entre la demanda potencial y el inventario de la empresa. Según el contenido del inventario se puede reajustar la demanda entre los competidores, en un proceso en el que intervienen más factores.
- Información interna. Por último se genera informes sobre todas las áreas de la empresa en función de los resultados obtenidos.

Cuando ya se han completado estos pasas para cada compañía, es cuando se generan los informes sobre el estado en el que queda el sistema para el siguiente ciclo.

Capítulo 3

Motivación y Objetivos

A continuación se va a describir cuáles son los principales motivos que han empujado a realizar este estudio, así como cuáles son las expectativas que se espera que este estudio resuelva.

Motivación

La empresa Simuladores Empresariales SL., formada a partir de un grupo de profesionales que habían colaborado en la Universidad Autónoma de Madrid, junto con un conjunto de profesores de la Universidad Carlos III de Madrid, construyó el simulador empresarial *SIMBA* (Simulation in Business Administration). Como hemos detallado SIMBA es un simulador enfocado a la formación de personas dedicadas a la toma de decisiones en empresas, que interactúan entre sí en un juego competitivo.

Uno de los mayores problemas que presenta SIMBA es que las fórmulas matemáticas que definen los procesos de simulación están implementadas dentro del código fuente del programa, y por tanto son muy costosas de modificar. Para empezar, cada modificación supone generar una nueva versión del sistema, con todas las tareas que eso implica. Además, las versiones anteriores son inválidas, ya que una modificación de una fórmula no supone, en general, una nueva funcionalidad, sino la corrección de un problema en la formulación. Los cambios son bastante frecuentes porque las reglas requieren frecuentes ajustes. Básicamente el problema de simulación que se presenta y en la manera que se trata de resolver es demasiado complejo para un planteamiento basado en un lenguaje imperativo. Como se puede observar el módulo de simulación no tiene las mismas necesidades que el resto del sistema, por lo que es importante identificar cuál es la mejor manera de trabajar en el módulo de simulación.

Una solución al problema concreto del módulo de simulación es orientar su diseño a un sistema basado en reglas. Recoger todo el conocimiento que se utiliza para realizar las simulaciones y estructurarlo en una representación basada en reglas. Utilizar un motor de reglas, permitiría ganar en flexibilidad; las reglas se podrían modificar individualmente, sin afectar al resto del sistema; acercar la implementación de la lógica a las personas que realmente conocen la lógica: en general las reglas utilizan un lenguaje bastante más

cercando al lenguaje natural que otras soluciones; simplificar la implementación de los algoritmos a los desarrolladores de productos: fragmentan la implementación de la solución y facilitan la implementación de estructuras lógicas;... Tratando de obtener estas mejoras se decidió utilizar la herramienta *CLIPS* para construir un simulador que funcionase exactamente igual que el simulador de SIMBA pero basado en reglas.

La introducción de CLIPS supuso importantes mejoras, sobre todo en lo que se refiere a independizar la parte que sufre mayor número de modificaciones (el simulador) del resto de la aplicación. Las reglas, al tener su propio ciclo de desarrollo, no afectan al resto del sistema al ser modificadas. El desarrollo llevado a cabo en CLIPS dio como resultado un conjunto de reglas que seguía siendo demasiado complejo para ser entendido por personas con bajos conocimiento en la herramienta.

CLIPS tiene una serie de deficiencias importantes, como pueden ser: la dificultad para gestionar las reglas de manera individualizada, la falta de herramientas que ayuden a escribir el código de las reglas, la dificultad de emplear mecanismos de coordinación entre varios miembros del equipo de desarrollo.... Si tenemos en cuenta que las tareas de desarrollo de sistemas complejos no suelen ser individuales, sino que suelen realizarse en equipos de múltiples personas y disciplinas es imprescindible poder escribir múltiples reglas en paralelo, utilizar mecanismos sencillos que visualicen qué trabajo se está realizando, poder recuperar de una manera sencilla y ágil un versión modificada de una regla, y éstas sólo son algunas de las funcionalidades más importantes que aporta un BRMS (Bussines Rules Managment System). Al no existir herramientas que se puedan integrar fácilmente con CLIPS y cubrir estas deficiencias puede ser bastante complicado trabajar con CLIPS en determinados entornos.

Una vez evaluada como positiva la utilización de un motor de reglas, es importante seguir avanzando en ésta línea y el siguiente paso lógico sería añadir las funcionalidades que forman parte de un BRMS y mejorarían el desarrollo del sistema de reglas y de las que carece CLIPS. A partir de aquí se abren dos opciones: llevar a cabo un desarrollo que cubra todos los requisitos de un BRMS o utilizar un software que ya cubra dichos requisitos y realizar una prueba de concepto que permita evaluarlos sobre el problema concreto que presenta la simulación realizada por SIMBA. Como hemos visto un BRMS completo tiene funcionalidades muy complejas, por eso, a primera vista es más interesante poder evaluar un BRMS ya existente.

En el planteamiento de evaluar un BRMS, se espera que se cumpla con los requisitos esenciales ya detallados, que esté licenciado como software libre, que esté suficientemente extendido como para tener la certeza de que su funcionamiento es bueno, y que exista abundante documentación sobre él. En este sentido *drools* es una de las mejores opciones.

Las razones por las cuales se ha decidido emplear drools son, para empezar su madurez. Actualmente se encuentre en la quinta versión estable es un proyecto muy maduro y su desarrollo está apoyado por una de las empresas más importantes en el mundo del software libre, RedHat. También se ha valorado de manera muy positiva, su orientación a entornos

profesionales, ya que cada evolución de drools ha ido marcada por la necesidad de suplir las necesidades de las organizaciones que han querido desarrollar productos que incluyen algún motor de reglas, incluso podemos decir que drools ha ayudado a acuñar el término BRMS. El hecho de que es un software libre, bien documentado por la comunidad de usuarios, que permite acceder al código fuente, también es valorable muy positivamente.

El planteamiento de construir un BRMS sería una buena idea, y tendría la ventaja de que determinados elementos, se podrían ajustar a las necesidades concretas del problema, siempre teniendo en cuenta las posibles evoluciones del mismo. Sin embargo, antes de abordar esta posibilidad parece más sencillo probar con un software contrastado que en caso de ajustarse perfectamente a nuestras necesidades no haría necesario el desarrollo de un BRMS. Si la evaluación no fuese lo suficientemente positiva, la experiencia de trabajar sobre una herramienta nos ayudaría a identificar de una manera más evidente los requisitos que deberíamos exigirle a un BRMS que desarrollásemos.

Objetivos

El objetivo del proyecto es evaluar la posibilidad de utilizar drools-guvnor como BRMS para ser empleado en construir un sistema de reglas que tenga la misma funcionalidad del simulador del sistema SIMBA, pero que aporte las ventajas al desarrollo de un BRMS. Gracias a trabajos realizados anteriormente [10], se sabe que es viable construir un simulador dinámico definiendo su comportamiento mediante el uso de reglas, así que no se trata sólo de evaluar la elaboración de las reglas utilizando las herramientas de drools, sino también ver cómo estas herramientas mejoran el trabajo gracias a las funcionalidades que ofrece la plataforma. Por tanto el principal objetivo es comprobar la *viabilidad* de drools para resolver el problema de la simulación, pero sobre todo *evaluar* si drools-guvnor cubre de manera satisfactoria los principales requisitos definidos para un BRMS y que estas funcionalidades supongan una mejora significativa del trabajo que se realiza a la hora de definir las reglas. Por último y no menos importante *comparar* la solución ofrecida por drools con la solución obtenida en el trabajo realizado en CLIPS.

Para realizar la tarea propuesta se utilizará la versión 5.1.1 de drools, publicada el día dos de agosto de 2010.

Viabilidad de la utilización de drools

El primer paso es comprobar si drools contiene todos los mecanismos necesarios para poder representar toda la lógica necesaria para cubrir las operaciones que realiza el simulador de SIMBA. Es decir, si el lenguaje de definición de reglas que utiliza drools, posibilita la representación de la lógica que requiere el módulo de arbitraje. No hay que olvidar que gran parte de las operaciones que realiza el módulo son operaciones matemáticas.

Para comprobar la viabilidad de drools, se llevará a cabo el desarrollo de las reglas necesarias para establecer los cálculos encargados de determinar el ranking de calidad de

las empresas que compiten, dentro del módulo de simulación de SIMBA. Las operaciones necesarias para elaborar éste ranking son suficientemente representativas del resto del simulador, por lo que implementar todo el módulo de simulación no aportaría más valor para certificar la viabilidad de la utilización de drools.

En la documentación sobre drools se especifica que no habría ningún problema en traducir la formulación matemática, que utiliza el simulador, en un conjunto de reglas escritas en drl (uno de los lenguajes de definición de reglas que emplea drools).

El objetivo no consiste simplemente en construir un conjunto de reglas que se resuelva el problema, sino intentar construir un conjunto de reglas que además de resolver el problema, lo haga de la forma más elegante posible. Para ello se va a realizar el esfuerzo necesario para seguir las recomendaciones de buenas prácticas que se encuentran distribuidas entre los manuales de drools. Algunas de las más importantes son:

- Mantener el mismo criterio dado por Java en cuanto a los tipos definidos para la información, nombrado de variables....
- Distinguir las variables colocando delante el carácter *$*, dólar.
- Las operaciones matemáticas se deben realizar en el consecuente.
- Limitar el tamaño de las reglas.
- No utilizar el operador *eval*, salvo que no exista otra forma de introducir la condición.
- Importar o incluir todos las entidades que forman parte del modelo, aunque no se usen directamente.

Seguir estas recomendaciones es importante porque pretenden mejorar la legibilidad del código de las reglas, prevenir errores cometidos comúnmente, y establecer un marco donde los desarrolladores de reglas puedan identificar fácilmente los elementos más importantes de las mismas.

Evaluación drools-guvnor.

Para construir las reglas se empleará drools-guvnor y dentro del estudio es muy importante evaluar si utilizar esta herramienta, supone una mejora significativa en el desarrollo ,y sobre todo, en el mantenimiento del módulo de simulación de SIMBA. Emplear un BRMS para llevar a cabo el desarrollo de un proyecto de reglas suele aparejar importantes beneficios, sin embargo, dado lo costoso que supone construir un aplicativo que reuna los requisitos de un BRMS y la existencia de drools-guvnor, se planteó como objetivo de este estudio comprobar si las funcionalidades que ofrece drools-guvnor cumplen con las especificaciones de lo que debe cubrir un BRMS, y si realmente suponen una mejora en el contexto en el que se desean emplear.

La documentación de drools-guvnor confirma que las funcionalidades más importantes que debe tener un BRMS están implementadas ([11]) en la versión que se va a evaluar. Las funcionalidades que se evaluarían en orden de importancia, para el problema concreto que nos atañe:

- **Edición de reglas**: facilidad a la hora de editar reglas, evaluar cómo de fácil sería crearlas para personas con pocos conocimientos en drools, capacidad de las reglas que se pueden crear.
- **Definición de la ontología**: herramienta para definir la ontología, impacto de definir la ontología en el aplicativo.
- **Gestión de versiones**: gestión de diferentes versiones de reglas.
- **Gestión de errores**: detección de errores sintácticos y semánticos. Detección de cambios que propagan errores.
- **Seguridad**: elementos protegidos, distinción entre usuarios. Sencillez de configuración.
- **Integración**: mecanismos facilitados para la integración con otros sistemas.

La evaluación se centrará en el problema concreto que supone construir las reglas para el simulador de SIMBA. Debido a la importancia de las operaciones matemáticas necesarias el mejor método para crear reglas es mediante el uso de drl, por eso no se evaluarán todos los posibles editores de reglas que incluye drools-guvnor y sólo se evaluará el editor guiado de reglas en drl.

Si atendemos a los aspectos que definen un BRMS no coinciden con los criterios de evaluación escogidos. La decisión se ha basado en escoger aquellos criterios que más valor aportan a la hora de solucionar el problema propuesto.

Un punto muy importante al margen del grado de satisfacción con el que se emplee la tecnología, sería tener en cuenta la diferencia entre emplear drools-guvnor y no emplear ningún BRMS.

Comparación con CLIPS.

Puesto que existe un punto de referencia con el que comparar el conjunto de reglas resultante con drools con uno desarrollado en CLIPS, no sería una comparación de tecnologías, más bien una comparación sobre los conjuntos de reglas resultantes. El objetivo sería realizar una comparación desde el punto de vista que ofrecería cada conjunto sobre la mantenibilidad (considerando esencialmente un mantenimiento evolutivo), la legibilidad (lo fácil difícil que es interpretar el funcionamiento del conjunto) y el nivel de conocimiento sobre la tecnología de los aspectos anteriores.

Capítulo 4

Memoria del trabajo

Una vez se han detallado la motivación que empuja éste estudio y los objetivos que se desean alcanzar, vamos a describir el trabajo realizado para poder alcanzarlos.

La herramienta más importante para el desempeño del estudio ha sido drools-guvnor, además se ha llevado a cabo una pequeña implementación sobre drools-expert para poder realizar pruebas y complementar el trabajo de manera satisfactoria.

4.1. Modelo de conocimiento

Para poder definir las reglas que se van a construir es necesario elaborar la ontología de la información que define la estructura de los datos sobre los que las reglas podrán operar. A esta ontología se le puede denominar modelo de conocimiento.

Para construir el modelo de conocimiento se llevó a cabo un análisis de cuales eran los datos de partida que se conocen y que son necesarios para resolver las fórmulas. Estos datos son los propios de las empresas, las decisiones que se han tomado para el ciclo actual de cálculo y las que se tomaron anteriormente pero siguen influyendo en la simulación del periodo presente. Como información de partida no sólo hay que contar con la información proveniente de la empresa, sino contar con una serie de valores que son empleados en diferentes cálculos. No hay que olvidar que la fuente de información para construir el sistema es un documento donde se especifican las fórmulas que emplea el simulador y los valores necesarios para que el simulador funcione, aunque en dicho documento haya valores que puedan ser interpretados como constantes, no se han tratado como tales buscando una solución con el menor número de limitaciones posibles.

Una vez determinada la información del problema que define su estado inicial (lo que es realmente la información que se almacena en la base de hechos), conviene analizar en profundidad las fórmulas empleadas tratando de descubrir si es conveniente emplear entidades para almacenar determinados resultados y darles de esa forma una estructura y un valor semántico. La pretensión de otorgarles un resultado semántico al margen del valor que arrojen las fórmulas es tratar de mejorar la compresión de la solución a través de los

resultados intermedios que ésta calcula.

En la figura (4.1), se puede observar la representación de la información en el modelo de conocimiento, las entidades que la forman y las relaciones entre las mismas. Las entidades que forman parte son: *CondicionMercado*, *DecisionCalidad*, *Empresa*, *Multiplicador* y *Parametro*. La relación entres las entidades es la siguiente, la entidad Empresa **toma** una DecisionCalidad y se ve **afectada** por un conjunto de entidades de tipo Multiplicador. Las entidades Parametro y CondicionMercado no están relacionadas con las otras entidades.

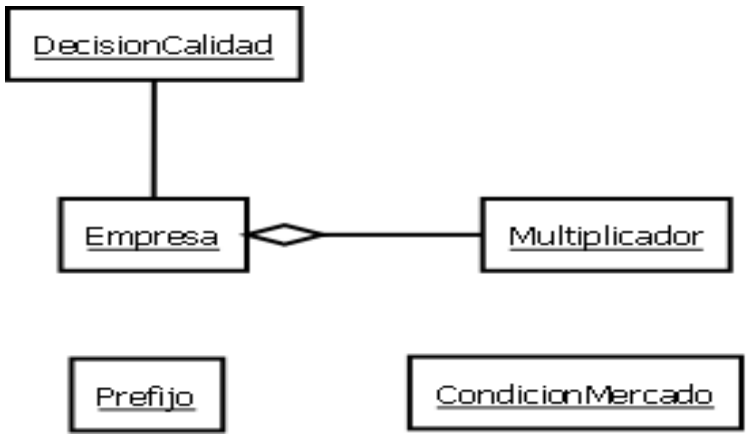


Figura 4.1: Relaciones entre las principales entidades del modelo de conocimiento.

Para definir el modelo de conocimiento dentro de la herramienta drools-guvnor, se ha empleado un formulario que guía al usuario en la creación de las diferentes entidades. Se ha escogido ésta forma para aprovechar al máximo las posibilidades de la plataforma. Sin embargo, una de las limitaciones impuestas por este procedimiento, es que el modelo no puede utilizar estructuras dinámicas de datos. Además las estructuras dinámicas nativas de datos de Java, que sí se pueden utilizar durante la creación de la ontología, no son explotables en el editor guiado de reglas, que es el editor que se desea emplear para escribir las reglas. Afortunadamente este inconveniente no genera un conflicto de viabilidad, pero obliga a introducir nuevas entidades que no aportan un valor significativo, salvo cuantificar las relaciones entre objetos del dominio cuando las relaciones son de uno a varios. En la figura 4.2, se puede ver como sería el modelo de conocimiento completo en el que se emplean éstas entidades auxiliares para resolver el problema de las relaciones múltiples.

Obviamente, los hechos que definen el estado del problema y las modificaciones que realicen las reglas sobre la memoria de trabajo deben ajustarse a la definición dada del modelo de conocimiento.

A continuación se realizará una pequeña descripción de las entidades del modelo y sus correspondientes atributos.

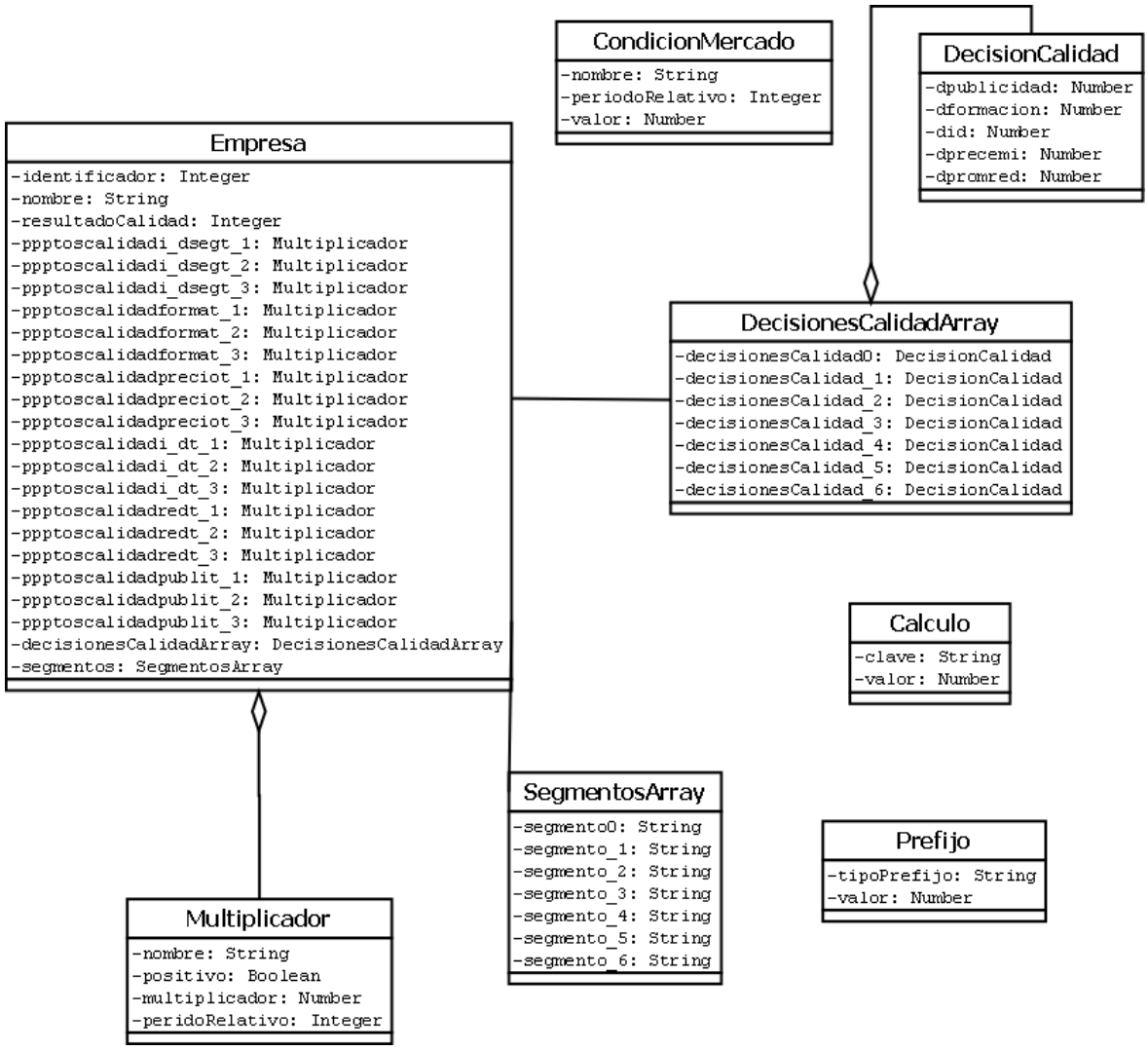


Figura 4.2: Diagrama UML de las entidades del modelo de conocimiento.

Empresa

Es la entidad sobre la que se realiza el cálculo de calidad. Almacena un histórico de las decisiones tomadas por las empresa que son pertinentes para los cálculos, así como los multiplicadores que afectan a la empresa en el periodo de cálculo.

Atributos	Tipo	Descripción
identificador	Número entero	Valor numérico por el que identifica unívocamente una empresa
nombre	Texto	Nombre por el que se conoce a la empresa

Continúa en la página siguiente

Atributos	Tipo	Descripción
resultadoCalidad	Número entero	Evaluación de calidad de la empresa para el periodo actual
ppptoscalidadi_dsegt_1	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la inversión I+D de las empresas del mismo segmento y la inversión I+D de todas las empresas, para el periodo actual.
ppptoscalidadi_dsegt_2	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la inversión I+D de las empresas del mismo segmento y la inversión I+D de todas las empresas, para el periodo anterior al actual.
ppptoscalidadi_dsegt_3	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la inversión I+D de las empresas del mismo segmento y la inversión I+D de todas las empresas, para dos periodos antes del actual.
ppptoscalidadformat_1	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la tasa de variación de presupuesto de la empresa en formación y la tasa de variación del mismo aspecto del resto de empresas, para el periodo actual.
ppptoscalidadformat_2	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la tasa de variación de presupuesto de la empresa en formación y la tasa de variación del mismo aspecto del resto de empresas, para el periodo anterior al actual.
ppptoscalidadformat_3	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la tasa de variación de presupuesto de la empresa en formación y la tasa de variación del mismo aspecto del resto de empresas, para dos periodos anteriores al actual.
ppptoscalidadpreciot_1	Multiplicador	Multiplicador para el cálculo del índice de calidad, que se decide en función de la relación entre el precio de una empresa del producto y el precio medio del segmento, para el periodo de tiempo actual.

Continúa en la página siguiente

Atributos	Tipo	Descripción
ppptoscalidadpreciot_2	Multiplicador	Multiplicador para el cálculo del índice de calidad, que se decide en función de la relación entre el precio de una empresa del producto y el precio medio del segmento, para el periodo de tiempo anterior al actual.
ppptoscalidadpreciot_3	Multiplicador	Multiplicador para el cálculo del índice de calidad, que se decide en función de la relación entre el precio de una empresa del producto y el precio medio del segmento, para el periodo de tiempo dos veces anterior al actual.
ppptoscalidadi_dt_1	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la tasa de variación de inversión de la empresa en I+D y la tasa de variación del mismo aspecto del resto de empresas, para el periodo actual.
ppptoscalidadi_dt_2	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la tasa de variación de inversión de la empresa en I+D y la tasa de variación del mismo aspecto del resto de empresas, para el periodo anterior.
ppptoscalidadi_dt_3	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la tasa de variación de inversión de la empresa en I+D y la tasa de variación del mismo aspecto del resto de empresas, para hace dos periodos.
ppptoscalidadredt_1	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre el presupuesto destinado a la red de ventas de una empresa y la media de este aspecto del presupuesto destinado por todas las empresas, para el periodo actual.
ppptoscalidadredt_2	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre el presupuesto destinado a la red de ventas de una empresa y la media de este aspecto del presupuesto destinado por todas las empresas, para el periodo anterior.

Continúa en la página siguiente

Atributos	Tipo	Descripción
ppptoscalidadredt_3	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre el presupuesto destinado a la red de ventas de una empresa y la media de este aspecto del presupuesto destinado por todas las empresas, para hace dos periodos.
ppptoscalidadpublit_1	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la inversión en publicidad de la empresa y la inversión media de empresas del mismo segmento, para el periodo actual.
ppptoscalidadpublit_2	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la inversión en publicidad de la empresa y la inversión media de empresas del mismo segmento, para el periodo anterior.
ppptoscalidadpublit_3	Multiplicador	Multiplicador que afecta a la empresa, según la relación entre la inversión en publicidad de la empresa y la inversión media de empresas del mismo segmento, para hace dos periodos.
decisionesCalidadArray	DecisionesCalidadArray	Decisiones que ha tomado la empresa sobre su presupuesto que afectan directamente al área de calidad, a lo largo de los periodos pertinentes para el cálculo de calidad.
segmentos	SegmentosArray	Segmentos a los que ha pertenecido la empresa en los últimos seis periodos.

CondicionMercado

Son los resultados de las fórmulas que se definen a partir de las decisiones que toman todas las empresas. Estos resultados son comunes a todas las empresas, que compiten con las mismas condiciones de mercado.

Atributos	Tipo	Descripción
nombre	Texto	Cadena de texto que identifica unívocamente a la condición.
periodoRelativo	Número entero	Periodo relativo sobre el que se ha realizado el cálculo.
valor	Número	Valor calculado para una condición.

DecisionCalidad

Esta entidad agrupa todas las decisiones relevantes de una empresa a la hora de medir su índice de calidad. Cada empresa tiene su propios valores de ésta entidad para cada periodo de los que se considera necesario para el cálculo.

Atributos	Tipo	Descripción
dpublicidad	Número	Inversión de la empresa realizada en publicidad.
dformacion	Número	Inversión de la empresa realizada en formación.
did	Número	Inversión de la empresa realizada en I+D.
dprecemi	Número	Precio medio.
dpromred	Número	Inversión de la empresa realizada en la red de ventas.

Multiplicador

Son valores que se aplicarán al cálculo del índice de calidad. Según las decisiones que tomase la empresa, en un periodo, respecto al global del conjunto de todas las empresas, se emplearán unos multiplicadores u otros, es decir, aunque los multiplicadores son constantes cada empresa emplea para calcular su rating de calidad unos distintos.

Atributos	Tipo	Descripción
nombre	Texto	Cadena de texto que identifica a un multiplicador.
positivo	Booleano	Indica si el multiplicador seleccionado es de naturaleza positiva o negativa.
multiplicador	Número	Valor que aplica el multiplicador a la fórmula.
peridoRelativo	Número entero	Periodo relativo sobre el que se ha realizado el cálculo

Prefijo

Son valores que se aplican a los cálculos de calidad.

Atributos	Tipo	Descripción
tipoPrefijo	Texto	Nombre por el que se conoce al prefijo.
valor	Número	Valor del prefijo.

Otros elementos del dominio

Además de los elementos descritos, existen otros elementos que no aportan tanta información sobre el dominio como los anteriores, pero que de alguna manera intervienen en la

solución del problema. Éstas se utilizan para cuantificar las relaciones o para realizar cálculos auxiliares.

DecisionesCalidadArray

Con la problemática de las estructuras dinámicas de datos, ésta ha sido una de las entidades necesarias. Esta entidad nos permitiría almacenar el histórico de las decisiones de calidad (representadas por DecisionCalidad), en los periodos anteriores, que son necesarias para realizar los cálculos que requiere el simulador.

Atributos	Tipo	Descripción
decisionesCalidad0	DecisionCalidad	DecisionCalidad para el periodo relativo 0 (periodo actual).
decisionesCalidad_1	DecisionCalidad	DecisionCalidad para el periodo relativo -1.
decisionesCalidad_2	DecisionCalidad	DecisionCalidad para el periodo relativo -2.
decisionesCalidad_3	DecisionCalidad	DecisionCalidad para el periodo relativo -3.
decisionesCalidad_4	DecisionCalidad	DecisionCalidad para el periodo relativo -4.
decisionesCalidad_5	DecisionCalidad	DecisionCalidad para el periodo relativo -5.
decisionesCalidad_6	DecisionCalidad	DecisionCalidad para el periodo relativo -6.

SegmentosArray

Debido al problema que existe con las estructuras dinámicas, para poder asociar los segmentos al que pertenece la empresa en cada periodo, se ha creado esta estructura.

Atributos	Tipo	Descripción
segmento0	Texto	Segmento al que pertenece la empresa en el periodo relativo 0 (periodo actual).
segmento_1	Texto	Segmento al que pertenece la empresa en el periodo relativo -1.
segmento_2	Texto	Segmento al que pertenece la empresa en el periodo relativo -2.
segmento_3	Texto	Segmento al que pertenece la empresa en el periodo relativo -3.
segmento_4	Texto	Segmento al que pertenece la empresa en el periodo relativo -4.
segmento_5	Texto	Segmento al que pertenece la empresa en el periodo relativo -5.
segmento_6	Texto	Segmento al que pertenece la empresa en el periodo relativo -6.

Calculo

Esta entidad es utilizada por las reglas, para poder almacenar el resultado de algunos cálculos intermedios que son reutilizados en varias reglas, pero que no tienen ningún significado semántico. Es auxiliar, por lo que la aplicación no tendría por qué tener conocimiento de la existencia de ésta entidad, ya que no es un hecho ni es un resultado del motor, sólo se gestiona a nivel interno de las reglas. Aún así para facilitar la depuración de errores de las reglas la aplicación interpreta esta entidad como una más.

Atributos	Tipo	Descripción
clave	Texto	Nombre por que el que se identifica al cálculo.
valor	Número	Resultado del cálculo.

4.2. Base de reglas

En este apartado se describirán las reglas creadas para poder calcular el índice de calidad de las empresas que participan en el simulador. Las reglas representan las fórmulas que se emplean dentro del módulo de arbitraje de SIMBA (Simulation in Business Administration).

Para la construcción de las reglas se ha puesto especial énfasis en que las reglas sean sencillas de comprender para las personas que posean conocimientos básicos en drools y/o en el dominio, que es una de las razones principales por las que se quiere trasladar la lógica de arbitraje a un sistema basado en reglas. Las consecuencias más visibles son que éstas tienen pocos antecedentes, y su consecuente sólo afecta a un único hecho de la memoria de trabajo (un único cálculo). Aunque no sea muy apreciable también se ha conseguido una gran independencia entre las reglas y un buen nivel de legibilidad.

Dentro de las reglas hay muy poca dependencia con los datos que maneja SIMBA. Un poco más en profundidad, se ha buscado la utilización de tipos de datos genéricos y en ninguna regla se ha presupuesto el valor de ningún valor por defecto, incluso cuando fuese justificable en el entorno del problema (tales datos, como por ejemplo, los segmentos a los que puede pertenecer una empresa).

El diseño de las reglas garantiza que sólo se realizan los cálculos y/o comprobaciones necesarios y no es necesario realizar comprobaciones extra para controlar el número de repeticiones de una regla o similares (como ocurre frecuentemente empleando tecnologías distintas a drools).

El sistema no está optimizado para resolver el problema en el menor tiempo posible. Se han seguido las consignas para no penalizar el rendimiento pero el principal objetivo ha sido primar la eficacia y el fácil entendimiento, por encima de la eficiencia. El gran problema del módulo de simulación es la mantenibilidad del sistema; la eficiencia en tiempo no parece un requisito muy importante ya que el cálculo del simulador no requiere una respuesta en tiempo real.

El número total de reglas desarrolladas asciende a *137* (dedicadas únicamente al módulo de calidad).

Aunque el proceso de ejecución de las reglas está dirigido por datos, interpretando el conjunto de reglas, se puede deducir que es necesario que algunas se ejecuten antes que otras, puesto que algunos antecedentes son añadidos por otras reglas a la memoria de trabajo. Según la funcionalidad de las reglas y en orden de ejecución las reglas se pueden agrupar en los siguientes conjuntos:

- **Cálculos preliminares.**
- **Cálculos de condiciones de mercado.**
- **Selección de multiplicadores.**
- **Cálculo de calidad.**

Figura 4.3: Ejemplo de regla en CLIPS empleada para suplir al simulador.

Consideraciones generales sobre las reglas escritas en CLIPS.

Puesto que existe una propuesta de implementación del módulo de arbitraje basada en CLIPS, vamos a realizar una pequeña comparativa entre algunas reglas escritas en drl para drools y las reglas que solventan la misma situación en CLIPS. Existe una serie de características que aparecen en la mayoría de las reglas escritas en CLIPS, para no repetirlos se señalarán ahora (4.3):

- Utilización de nombres poco significativos (1). En algunos elementos de la ontología con la que se han definido las reglas se han empleado nombres poco descriptivos (en especial para los atributos), de la misma forma no se ha empleado nombres descriptivos para los títulos de reglas. El problema es que cuesta asociar determinados elementos del dominio en las reglas en las que aparece.
- Asunción de valores en el interior de las reglas (2). En concreto se ha asumido que habrá únicamente 3 segmentos y que sus valores serán “A”, “M” y “B”. El principal problema es que tal y como se ha construido el sistema de reglas en

CLIPS la aparición de un nuevo segmento supondría la modificación de gran parte de las reglas que forman parte del sistema de calidad.

- Controlar el estado en el que se encuentra la aplicación (3). Se añade información que no es importante para el problema, aunque pueda ser imprescindible para el motor pero que no es significativo en la solución del problema. Además al emplear esta información como antecedente en todas las reglas se fuerza una ejecución secuencial de dichas reglas que va en contra de la filosofía de un sistema basado en reglas.

Cálculos preliminares

En el dominio, para calcular las condiciones de mercado, es necesario sumar algún elemento de las empresas, o de las empresas de un mismo segmento, para un periodo dado (por ejemplo sumar todas las inversiones en I+D de todas las empresas en el periodo actual). La mayoría de estos sumatorios se repiten para realizar el cálculo de alguna condición del mercado en distintos periodos, incluso en distintas condiciones de mercado. Para evitar que estos cálculos se realicen continuamente en el sistema, se han creado reglas que realice cada sumatorio que sea necesario una única vez, y añada a la memoria de trabajo el resultado del sumartorio y así pueda ser reutilizado las veces que sea preciso. Además, gracias a emplear estas reglas, se ha conseguido el efecto lateral de simplificar las reglas que resuelven las fórmulas relacionadas con las condiciones de mercado.

Como ya se ha comentado anteriormente no hay asunción de ningún tipo sobre la naturaleza de los cálculos, es decir no se está limitando ni los valores que pueden tomar los segmentos, ni los identificadores que pueden tomar las empresas.

La figura 4.4 es un ejemplo de una regla de cálculo preliminar. Puesto que este cálculo se realiza para todas las emprsas de un mismo segmento, en la línea 4, se selecciona sobre que segmento se va a realizar el cálculo. En la línea 5 se comprueba que este cálculo no se ha realizado todavía. El elemento más destacado de la sintaxis es la sentencia *from* (línea 6). Como ya se ha comentado 2.3, esta sentencia nos permite realizar operaciones sobre un conjunto de datos, especificando una condición y, por supuesto, una operación. En ocasiones la operación puede ser booleana, y lo que se buscaría es conseguir una condición de activación de la regla, sin embargo en este caso no es así al ser una operación matemática. El resultado del operador se almacena en una variable, para poder ser empleada en el consecuente (aunque se podría emplear en el antecedente). Por definición de la sintaxis soportada, este tipo de expresiones sólo puede usarse en el antecedente. Una vez realizado el cálculo se crea un elemento de la entidad Calculo para guardar el resultado de la operación (líneas 9-12).

En el consecuente se añade una entidad de tipo cálculo, que será antecedente en las reglas que necesiten este cálculo para activarse. Este hecho no tiene un valor significativo en el dominio.

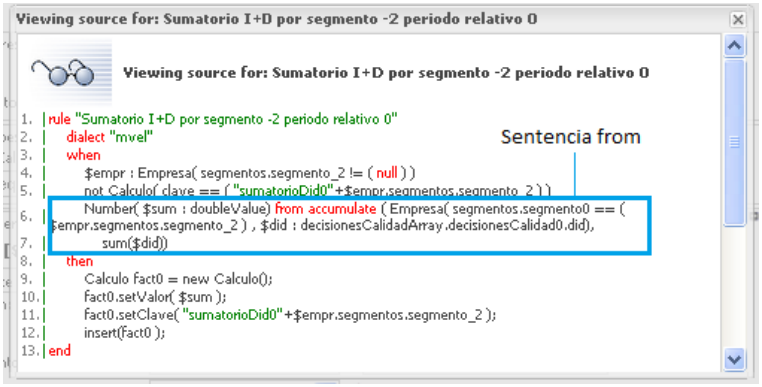


Figura 4.4: Ejemplo de regla de cálculo preliminar.

Condiciones mercado

Una vez que se han realizado los cálculos preliminares, el siguiente paso es calcular las condiciones de mercado. Las condiciones de mercado suelen involucrar varios sumatorios, que se realizan con las reglas anteriores. Las reglas referentes a las condiciones de mercado incluyen uno de estos tres tipos de operaciones: **acumulado**, **tasa de variación** o **promedio**.

Una fórmula típica de una condición de mercado que se genera mediante un acumulado se puede observar en la figura 4.5. En este caso es un sumatorio de una decisión que toman las empresas en un periodo de tiempo determinado. En la figura 4.6 se ve cómo se ha representado esta fórmula en CLIPS. Para esta regla se han llevado a cabo previamente un acumulado por segmento habiendo sumado previamente las decisiones de todas las empresas que pertenecen al mismo segmento por periodo, de ésta manera se ha simplificado este cálculo. Una filosofía muy similar a la empleada para desarrollar ésta regla en drools (figura 4.7), aunque aparentemente esta regla parece más sencilla y legible. En la regla escrita en drools, se recuperan los hechos necesarios que representan los cálculos preliminares calculados previamente (líneas 5 a 8), se comprueba que no se ha realizado el cálculo todavía (9) y en el consecuente se crea el hecho que representa el resultado y se añade a la memoria de trabajo (11 a 15).

$$presuFormAcumMer(-4) = \sum_{t=-1}^{-4} \sum_{x=A}^{F} empresa(x).DFORMACION(t)$$

Figura 4.5: Ejemplo de fórmula de condiciones de mercado, cálculo acumulado (Cálculo del presupuesto de formación acumulado para t -1). Esta fórmula representa el sumatorio de todas las inversiones de formación de todas las empresas desde el ciclo anterioa al actual hasta cuatro anteriores.

Las reglas de tasa de variación resuelven fórmulas como la que se puede observar en la figura 4.8, que calcula la variación del acumulado de un determinado aspecto del presupuesto de las empresas, entre dos periodos adyacentes. En la figura 4.9 podemos ver cómo se ha resuelto esta fórmula en el sistema basado en drools y en 4.10 como se ha resuelto empleando CLIPS. En la regla de CLIPS, se ha realizado éste cálculo para todos

```
(defrule Cal-011X
    (declare (salience 464))
    (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
    ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po) (ESTADOCAL 41)
        (CAL011A0 ?cal011A0) (CAL011B0 ?cal011B0) (CAL011C0 ?cal011C0)
        (CAL011A_1 ?cal011A_1) (CAL011B_1 ?cal011B_1) (CAL011C_1 ?cal011C_1)
        (CAL011A_2 ?cal011A_2) (CAL011B_2 ?cal011B_2) (CAL011C_2 ?cal011C_2)
        (CAL011A_3 ?cal011A_3) (CAL011B_3 ?cal011B_3) (CAL011C_3 ?cal011C_3))
    (contadorcal (cequipos ?ideo))
        =>
    (modify-instance ?res (CAL011X (+ ?cal011A0 ?cal011B0 ?cal011C0 ?cal011A_1 ?cal011B_1 ?cal011C_1
        ?cal011A_2 ?cal011B_2 ?cal011C_2 ?cal011A_3 ?cal011B_3 ?cal011C_3)) (ESTADOCAL 42))
)
```

Figura 4.6: Versión de CLIPS de la fórmula: Cálculo del presupuesto de formación acumulado para t -1.

los periodos que son necesarios, algo, que según las recomendaciones de drools, no se considera una práctica idónea, siguiendo las recomendaciones de drools. En la regla escrita en drools, simplemente se recupera la empresa de la que se desea realizar el cálculo (línea 4), se realiza la operación (6) y se actualiza la memoria de trabajo (7). La regla en CLIPS es muy similar, sólo que reliza tres veces la operación. Reglas como ésta que sólo afecta a una empresa y no requieren ningún tipo de cálculo previo, pueden ser seleccionadas en cualquier momento por el motor de reglas en el caso de drools, sin embargo se ha incluido aquí debido a su similitud con las reglas de tasa de variación. En el caso de CLIPS al tener un estado en el que que se deben ejecutar las reglas esta regla solo podría ejecutar en el momento en el las empresas alcancen el estado fijado.

Otro grupo de reglas que se emplean para el cálculo de las condiciones de mercado son las que incluyen operaciones de promedios. Estas reglas resuelven fórmulas como la que viene representada en la figura 4.11. Lo que se realiza en estas fórmulas es sumar una la decisión que tomó una empresa sobre un aspecto en un determinado periodo. Además algunas fórmulas (como es 4.11) incluyen la limitación que toda las empresas de la suma deben de pertenecer al mismo segmento. Comparando cómo se resuelve la fórmula en drools (figura 4.12) y en CLIPS (figura 4.13), podemos comprobar que en CLIPS es necesario utilizar tres reglas para poder satisfacer la fórmula, además las operaciones son mucho más difíciles de interpretar. La regla en drools es más fácilmente interpretable.

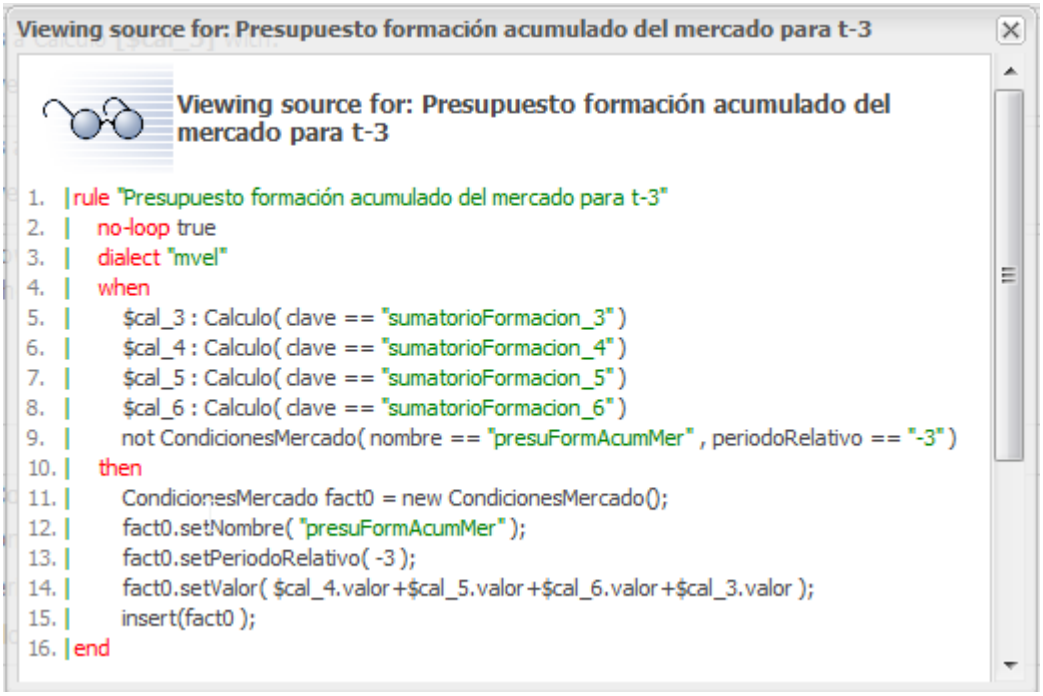


Figura 4.7: Versión de la regla en drools de la fórmula: Cálculo del presupuesto de formación acumulado para t -1.

$$empresa(x).varForm(-2) = (empresa(x).DFORMACION(-2) + empresa(x).DFORMACION(-3))/empresa(x).DFORMACION(-3)$$

Figura 4.8: Ejemplo de fórmula de condiciones de mercado, tasa de variación (Tasa de variación de presupuesto en formación de la empresa del t-2 respecto t-3). Dada una empresa calcula el cuál es el diferencial relativo de la inversión en formación entre dos periodos contiguos (el periodo dos veces anterior al actual y el anterior a éste).

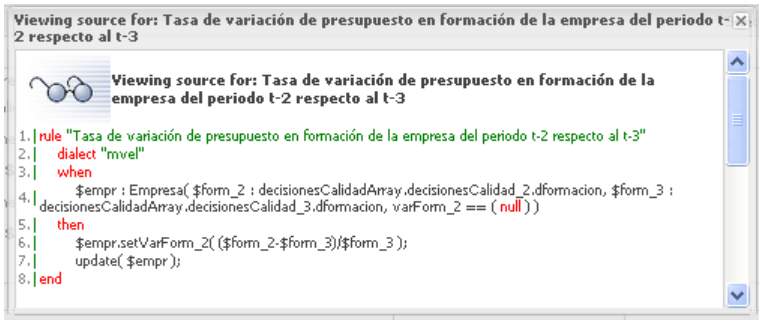


Figura 4.9: Versión de drools de la fórmula: Tasa de variación de presupuesto en formación de la empresa del t-2 respecto t-3.

```
(defrule Cal-012
    (declare (salience 462))
    (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
    (object (is-a HISTORIALDECISIONES) (IDEMPRESA ?ideo) (PERIODO ?po) (DFORMACION ?dformacion0))
    (object (is-a HISTORIALDECISIONES) (IDEMPRESA ?ideo) (PERIODO ?p_1) (DFORMACION ?dformacion_1))
    (object (is-a HISTORIALDECISIONES) (IDEMPRESA ?ideo) (PERIODO ?p_2) (DFORMACION ?dformacion_2))
    (object (is-a HISTORIALDECISIONES) (IDEMPRESA ?ideo) (PERIODO ?p_3) (DFORMACION ?dformacion_3))
    ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po) (ESTADOCAL 44))
    (contadorcal (cequipos ?ideo))
    (test (eq (- ?po ?p_1) 1))
    (test (eq (- ?p_1 ?p_2) 1))
    (test (eq (- ?p_2 ?p_3) 1))
        =>
    (modify-instance ?res (ESTADOCAL 45)
        (CAL012W (* (/ (- ?dformacion0 ?dformacion_1) ?dformacion_1) 100))
        (CAL012X (* (/ (- ?dformacion_1 ?dformacion_2) ?dformacion_2) 100))
        (CAL012Y (* (/ (- ?dformacion_2 ?dformacion_3) ?dformacion_3) 100)))
)
```

Figura 4.10: Versión de CLIPS de la fórmula: Tasa de variación de presupuesto en formación de la empresa.

$$precioMedioSeg(-2,S) = \sum_{x=Abase_{soloparaempresasdelsegmentoS}}^{F} empresa(x).DPRECIOEMP(-2)$$

Figura 4.11: Ejemplo de fórmula de cálculo de promedio, tasa de variación (Precio Medio Segmento para t-2). Esta regla suma todos los precios medios de las empresas que pertenecen a un mismo segmento.

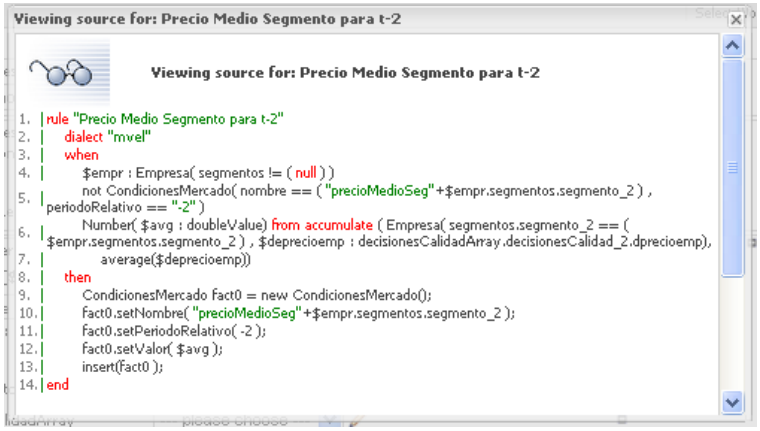


Figura 4.12: Ejemplo de regla de condiciones de mercado, cálculo de promedio.

```
(defrule Cal-006X_A     (declare (salience 474))
    (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
    (object (is-a HISTORIALAREACOMERCIAL) (IDEMPRESA ?ideo) (PERIODO ?pd) (RSEGMENTO ``A''))
    ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po) (ESTADOCAL 26)
        (CAL006A_1 ?cal006A_1) (CAL006A_1CONT ?cal006A_1cont))
    (contadorcal (cequipos ?ideo))
    (test (= (- ?po ?pd) 2))
        =>
    (modify-instance ?res (CAL006X (/ ?cal006A_1 ?cal006A_1cont)) (ESTADOCAL 27))
)
(defrule Cal-006X_B
     (declare (salience 474))
    (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
    (object (is-a HISTORIALAREACOMERCIAL) (IDEMPRESA ?ideo) (PERIODO ?pd) (RSEGMENTO ``M''))
    ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po) (ESTADOCAL 26)
        (CAL006B_1 ?cal006B_1) (CAL006B_1CONT ?cal006B_1cont))
    (contadorcal (cequipos ?ideo))
    (test (= (- ?po ?pd) 2))
        =>
    (modify-instance ?res (CAL006X (/ ?cal006B_1 ?cal006B_1cont)) (ESTADOCAL 27))
)
(defrule Cal-006X_C
    (declare (salience 474))
    (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
    (object (is-a HISTORIALAREACOMERCIAL) (IDEMPRESA ?ideo) (PERIODO ?pd) (RSEGMENTO ?rs&:(or(eq ?rs ``B'')(eq ?rs
``N''))))
    ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po) (ESTADOCAL 26)
        (CAL006C_1 ?cal006C_1) (CAL006C_1CONT ?cal006C_1cont))
    (contadorcal (cequipos ?ideo))
    (test (= (- ?po ?pd) 2))
        =>
    (modify-instance ?res (CAL006X (/ ?cal006C_1 ?cal006C_1cont)) (ESTADOCAL 27))
)
```

Figura 4.13: Versión de CLIPS de la fórmula: Precio Medio Segmento para t-2.

La regla escrita en drools de ejemplo (4.12), busca un segmento sobre el que falte por calcular el valor de su promedio (líneas 4 y 5), si existe alguno, realiza el sumatorio de los precios medios para el periodo de todas las empresas del segmento seleccionado (6). En el consecuente crea la condición de mercado y le asigna los valores necesarios (9-12) e insertar el nuevo hecho en la memoria de trabajo (13).

En todas las reglas de condiciones de mercado, que no son por segmento, se añade el atributo *no-loop*, que garantiza que cada regla sólo se activa una vez. Esto es debido a que estos cálculos sólo se necesitan calcular una vez y que es la forma más limpia de evitar que las reglas se activen continuamente, la alternativa sería incluir la condición de que el hecho

no existe. Las reglas que calculan una determinada condición de mercado para un segmento, se repiten tantas veces como segmentos haya, incluyendo una condición de que el resultado del cálculo exista o no para evitar que se ejecute múltiples veces la regla.

Selección de multiplicadores

Una vez se han establecido las condiciones de mercado, el siguiente paso consiste en seleccionar los multiplicadores que afectan a cada empresa. Cada condición de mercado en un periodo de tiempo se compara con la decisión que tomó la empresa referente a la condición de mercado y del mismo periodo. Si la comparación resulta favorable para la empresa, el multiplicador seleccionado será el positivo. En cada caso de que la comparación sea desfavorable para la empresa el multiplicador escogido será el negativo. Un ejemplo de éstas fórmulas se puede observar en la figura (4.14).

Salvo los puntos inicialmente comentados, tanto la versión en CLIPS (figura 4.15), como la versión de drools 4.16 son muy similares. Sobre la representación de la fórmula que se hace en drools habrá una regla que se active si el multiplicador a activar es positivo (la regla de ejemplo el multiplicador seleccionado es el negativo) y otra si es negativo, las condiciones de estas reglas son mutuamente excluyentes. Una diferencia con la forma de trabajar del conjunto de reglas escrito en CLIPS, es que ésto no es así y no existe otra versión de la regla que defina cuando se aplica el multiplicador negativo. Los pasos que sigue la regla en drools expuesta de ejemplo, son:

- □ Seleccionar el valor de la decisión de la empresa que se va a comparar (línea 4).
- □ Comprobar la existencia de la condición de mercado contra la que se va a realizar la comparación y realizar una de las comparaciones (5 y 6).
- □ Seleccionar el multiplicador que se va a añadir en el consecuente (el nombre y el signo) (7).
- □ En el consecuente, simplemente asignar el multiplicador seleccionado a la empresa con la que se había activado la regla (9 y 10).

$$kol = (empresa(x).varSegmentoID(0) > varMercadoID(0))?ppptoscalidadi\text{+}dsegt\text{-}2(positivo) : ppptoscalidadi\text{+}dsegt\text{-}2(negativo)$$

Figura 4.14: Ejemplo de fórmula de selección de multiplicador (Selección de multiplicador kol). En este ejemplo se evalúa la condición si para una empresa dada la variación de inversión en I+D es mayor que la variación de la inversión en I+D de todo el mercado. En caso de ser cierto utiliza como multiplicador kol el valor positivo de la constante ppptoscalidadi+dsegt_2, si no es cierto emplea el negativo.

```
(defrule Deci-007
    (declare (salience 458))
    (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
    ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po) (CAL004X ?calc4) (CAL005X ?calc5))
    ?dec <- (object (is-a DECISIONES))
    (contadorcal (cequipos ?ideo))
    (test (>= ?calc4 (- ?calc5 0.0000000000001)))
        =>
    (modify-instance ?dec (DECI007 0.3))
)
```

Figura 4.15: Versión de CLIPS de la fórmula: selección de multiplicador kol (parte positiva).

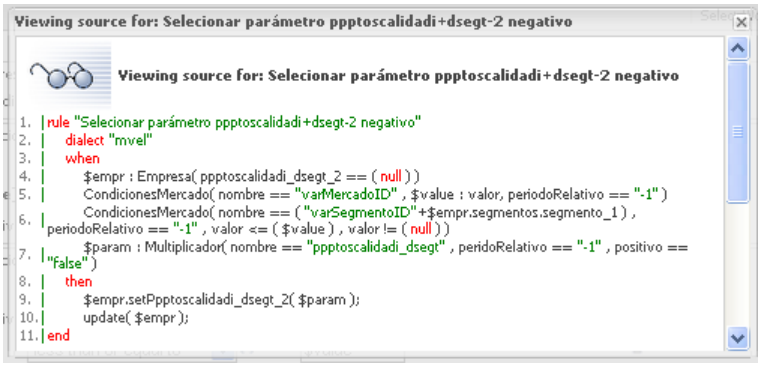


Figura 4.16: Versión de drools de la fórmula: Selección de multiplicador kol negativo.

Cálculo de calidad

Cuando ya se han seleccionado todos los multiplicadores, ya tenemos todos los elementos para realizar el cálculo para establecer el ranking de calidad (4.17). La fórmula final utiliza los mencionados multiplicadores, los prefijos y los resultados anteriores de calidad que obtuvieron las empresas. Es una fórmula bastante compleja que utiliza muchos elementos, esta característica se traslada a su traducción en CLIPS (4.18) y a la regla escrita en drl (4.19).

$$RCALI = (a * ppcalidadt\text{-}1) + (b * ppcalidadt\text{-}2) + (cod * ppcalidadi\text{+}d) + (eof * ppcalidadi\text{+}d) + (goh * ppcalidadi\text{+}d) + (ioj * ppcalidadi\text{+}dseg) + (kol * ppcalidadi\text{+}dseg) + (mon * ppcalidadi\text{+}dseg) + (oop * ppcalidadprecio) + (kor * ppcalidadprecio) + (sot * ppcalidadprecio) + (uov * ppcalidadprecio) + (wox * ppcalidadpubli) + (yoz * ppcalidadpubli) + (aaoab * ppcalidadpubli) + (acoad * ppcalidadred) + (aeoaf * ppcalidadred) + (agoah * ppcalidadred) + (aioaj * ppcalidadforma) + (akoal * ppcalidadforma)$$

Figura 4.17: Fórmula para calcular el rating de calidad de una empresa. Para cada empresa, multiplica los multiplicadores se han seleccionado por el prefijo correspondiente y suma los resultados.

En el consecuente se comprueba que todos y cada uno de los multiplicadores de cada se han seleccionado y en el antecedente se modifica la empresa para asignarle el resultado de la fórmula de calidad, como se puede observar es una fórmula bastante compleja.

```
(defrule Cal-014
    (declare (salience 456))
    (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
    (object (is-a HISTORIALAREACOMERCIAL) (IDEMPRESA ?ideo) (PERIODO ?pd) (RCALI ?rcali_1))
    (object (is-a HISTORIALAREACOMERCIAL) (IDEMPRESA ?ideo) (PERIODO ?pe) (RCALI ?rcali_2))
    ?dec <- (object (is-a DECISIONES) (DECI003 ?dec003) (DECI004 ?dec004) (DECI005 ?dec005) (DECI006 ?dec006)
        (DECI007 ?dec007) (DECI008 ?dec008) (DECI009 ?dec009) (DECI010 ?dec010) (DECI011 ?dec011) (DECI012 ?dec012)
        (DECI013 ?dec013) (DECI014 ?dec014) (DECI015 ?dec015) (DECI016 ?dec016) (DECI017 ?dec017) (DECI018 ?dec018)
        (DECI019 ?dec019) (DECI020 ?dec020))
    ?pre <- (object (is-a PREFIJOS) (PPCALIDADT-1 ?ppcalidadt_1) (PPCALIDADT-2 ?ppcalidadt_2) (PPCALIDADI+D
?ppcalidadid)
        (PPCALIDADI+DSEG ?ppcalidadidseg) (PPCALIDADPRECIO ?ppcalidadprecio) (PPCALIDADPUBLI ?ppcalidadpubli)
        (PPCALIDADRED ?ppcalidadred) (PPCALIDADFORMA ?ppcalidadforma))
    ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po) (ESTADOCAL 46))
    (contadorcal (cequipos ?ideo))
    (test (= (- ?po ?pd) 1))
    (test (= (- ?po ?pe) 2))
        =>
    (modify-instance ?res (CAL014 (+ (* ?ppcalidadt_1 ?rcali_1) (* ?ppcalidadt_2 ?rcali_2) (* ?ppcalidadid ?dec003)
        (* ?ppcalidadid ?dec004) (* ?ppcalidadid ?dec005) (* ?ppcalidadidseg ?dec006) (* ?ppcalidadidseg ?dec007)
        (* ?ppcalidadidseg ?dec008) (* ?ppcalidadprecio ?dec009) (* ?ppcalidadprecio ?dec010) (* ?ppcalidadprecio
?dec011)
        (* ?ppcalidadpubli ?dec012) (* ?ppcalidadpubli ?dec013) (* ?ppcalidadpubli ?dec014) (* ?ppcalidadred ?dec015)
        (* ?ppcalidadred ?dec016) (* ?ppcalidadred ?dec017) (* ?ppcalidadforma ?dec018) (* ?ppcalidadforma ?dec019)
        (* ?ppcalidadforma ?dec020))) (ESTADOCAL 47))
)
```

Figura 4.18: Versión de CLIPS de la fórmula: Cálculo del rating de calidad.

Con reglas cómo las descritas se cubre las necesidades que plantea la parte que evalúa la calidad de las empresas, dentro del simulador SIMBA. Se ha mostrado además una completa comparación entre las soluciones construidas en drools y en CLIPS. Un elemento muy importante relacionado con la creación de reglas es drools-guvnor, de la experiencia de uso con esta herramiente se hablará, en la sección 5.

Figura 4.19: Regla del índice de calidad.

4.3. Arquitectura del sistema

Una vez se han desarrollado las reglas, el siguiente paso consiste en ejecutar las reglas que se han desarrollado en un entorno externo a drools-guvnor pero que se integre perfectamente. El objetivo es plantear que drools-guvnor es un entorno de desarrollo y crear un pequeño entorno externo, que cumpla el rol de un entorno de ejecución del motor de inferencia. Es importante describir cómo interacciona todo el sistema entre sí. En un sistema basado en reglas (4.20) existen tres elementos fundamentales, la base de reglas, la base de hechos y el motor de inferencia. El primer elemento es el propio drools-guvnor, que es donde se van a almacenar las reglas una vez creadas. La base de hechos es la fuente de datos donde se almacena el estado de los hechos en el momento inicial, esta parte es la menos relevante en este estudio. Respecto al motor de inferencia se ha realizado un pequeño desarrollo que es un cobertor sobre drools-expert que se encarga de recuperar las reglas escritas en drools-guvnor, cargar los datos y darle toda esa información a drools-expert.

Más allá de testear la integración de drools-guvnor, construir este sistema permitirá poder realizar pruebas en un entorno no dirigido por drools-guvnor, muy importante dadas las deficiencias con las que cuenta el entorno de pruebas del propio drools-guvnor; y comprobar como se integra drools-guvnor con otro sistema (característica fundamental de un sistema gestor de reglas de negocio, BRMS).

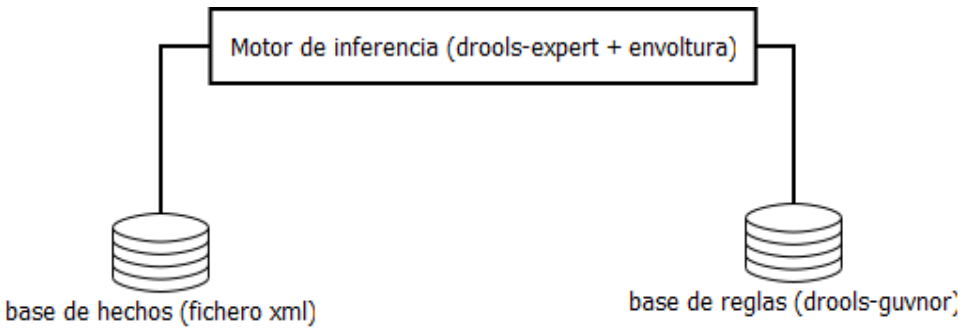


Figura 4.20: Arquitectura de la implementación del motor de reglas.

Base de reglas

La principal función de drools-guvnor dentro de este sistema es la de servir de base de reglas. Aunque no sea su principal funcionalidad, es necesario poder evaluar los mecanismos de integración de los que provee.

Es importante recalcar que para realizar las pruebas no se ha llevado a cabo ninguna modificación, ni de la configuración por defecto, ni de su código fuente.

Para su integración con los otros módulos es muy importante tener en cuenta las siguientes características de drools-guvnor:

- Para acceder a las reglas, existirá un servicio web que al ser invocado devolverá el conjunto de reglas. Este servicio se podrá parametrizar para obtener diferentes versiones etiquetadas del conjunto de reglas.
- El modelo de datos será parte del conjunto de reglas. Aunque no es una exigencia de drools-guvnor, se hará así para probar esta funcionalidad dentro del estudio.
- Al recuperar las reglas, éstas estarán compiladas (sin errores de sintaxis).
- El acceso a la url para obtener las reglas será bajo autenticación.

Base de hechos

En el entorno en el que se ejecuta SIMBA, la información que se utilizaría como base de hechos se almacena en una base de datos. Respecto a las pruebas que se desean realizar es irrelevante cuál es el soporte sobre el que se almacenan los hechos. Para simplificar esta necesidad se ha decidido que la base de hechos sea simplemente un fichero en formato xml con la información necesaria.

Se ha escogido xml, porque es un formato muy empleado, sencillo de implementar su procesamiento, fácilmente reconocible, y existen multitud de herramientas para trabajar con ficheros en este formato, en entorno jee, entre otros motivos.

Motor de inferencia

Su responsabilidad es cargar las reglas almacenadas en el BRMS, recuperar la información que almacena la base de hechos y ejecutar el motor de inferencia.

El motor de reglas está implementado (se va a emplear drools-expert), pero es necesario acceder a la base de reglas, cargarlas en el motor; leer la base de hechos y estructurar la información de tal manera que el motor las pueda interpretar correctamente para cruzarlas con las reglas que se van a emplear. Para poder complementar todas las funcionalidades es necesario complementar drools-expert con un pequeño desarrollo en Java.

Se han incluido algunas funcionalidades para automatizar la comprobación de si los resultados que obtiene el motor al alcanzar la condición de parada son similares al resultado esperado.

Anteriormente se comentó que drools ofrece la oportunidad de escribir el modelo mediante clases Java o en un lenguaje propio 2.3, para aprovechar mejor las posibilidades de drools-guvnor el modelo se ha generado utilizando el lenguaje propio de drools. Como consecuencia de ésto, el esquema de datos que se emplea para leer la base de hechos es distinto del que tiene la base de reglas, con lo que antes de insertar los hechos que se leen en el motor es necesario transformar los datos de la base de hechos para que las reglas puedan procesar los datos.

El diagrama de clases del programa se puede consultar en la figura (4.21).

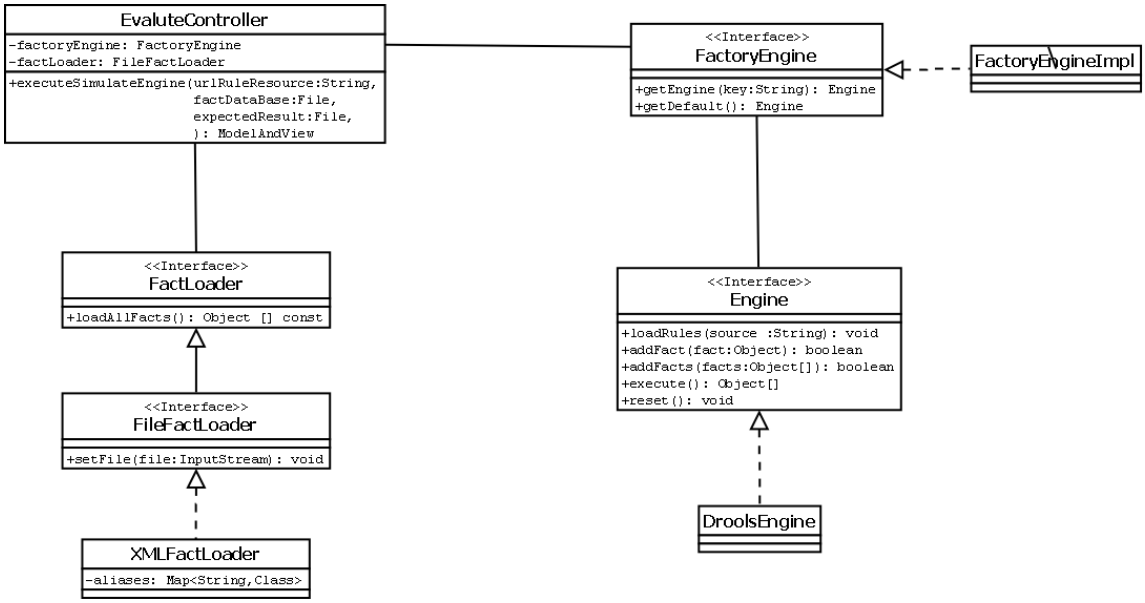


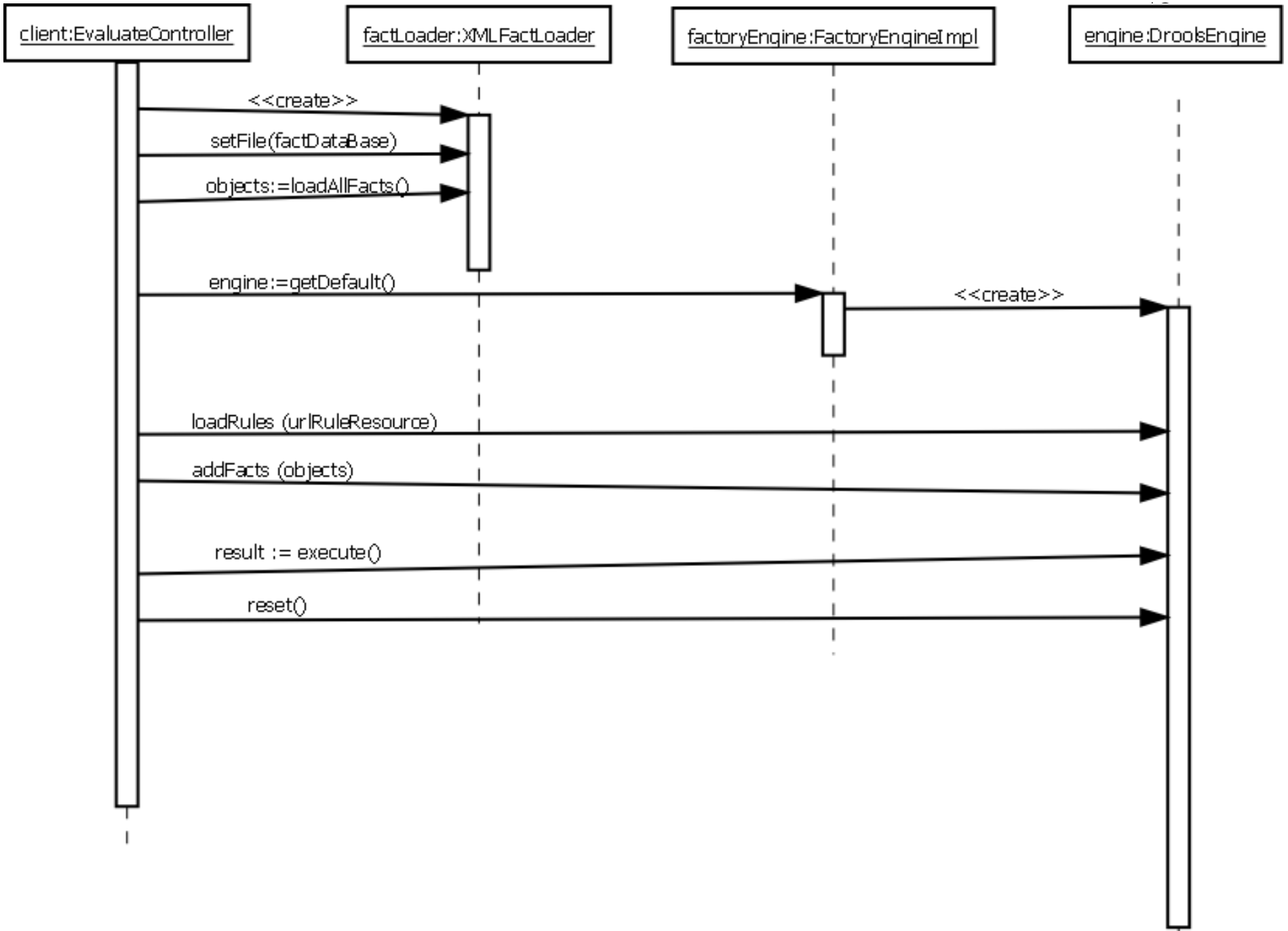
Figura 4.21: Diagrama UML de clases sin incluir las clases que representan los hechos.

Las interfaces **FactLoader** y **FileFactLoader** definen cómo deben cargarse los hechos de la base de hechos. La primera hace referencia a cualquier fuente de hechos, mientras que la segunda debe implementarse en caso de que la fuente de hechos sea un fichero, como es el caso que se presenta. La implementación concreta es **XMLFaclLoader**, que es capaz de cargar los hechos de un fichero XML.

La interfaz **FactoryEngine** describe los métodos de un cargador de motores de inferencia. La implementación que se va a emplear en este caso **FactoryEngineImpl** se basa en una configuración concreta para saber qué motor cargar.

Engine es una interfaz que define las operaciones que debe soportar un motor de inferencia. Se ha llevado a cabo una implementación (**DroolsEngine**) que es un wrapper sobre drools-expert.

Para tener una visión completa de cómo funciona este subsistema es bastante interesante ver el diagrama de secuencia (4.22).


client:EvaluateController
factLoader:XMLFactLoader
factoryEngine:FactoryEngineImpl
engine:DroolsEngine
<<create>>
setFile(factDataBase)
objects:=loadAllFacts()
engine:=getDefault()
<<create>>
loadRules (urlRuleResource)
addFacts (objects)
result := execute()
reset()

El cliente crea el cargador de hechos. Puesto que lo que se trata de llevar a cabo es una simulación, sí que existe la necesidad de tener una dependencia entre el cliente y el cargador de hechos (en concreto la clase XMLFactLoader). Una vez creada la clase, se le indica cuál es el fichero que tiene la información organizada en forma de objetos serializados en formato XML. Una vez que se lee la información a través de la clase FactoryEngineImpl, se crea la clase DroolsEnginge, que encapsula la invocación del motor de drools-expert. Utilizar una clase a modo de factoría que nos proteja de nuestra dependencia con el motor de reglas nos permitiría realizar las mismas simulaciones con otro motor de reglas simplemente implementando una clase que hiciese las veces de fachada y que implementase la interfaz Engine. A través de la clase DroolsEngine se cargan los hechos y se ejecuta el motor. Una vez que se ha ejecutado el motor se recuperan todos los objetos pertenecientes a la memoria de trabajo del motor. Puesto que el cliente además evalúa el resultado, se compara el resultado obtenido (los objetos de la memoria de trabajo) con el esperado y se muestra el resultado de los objetos que se esperan y de los que se reciben. Esta sería la forma en la que la aplicación realiza las simulaciones y las evalúa.

Uno de los objetivos que se ha pretendido es que sea posible utilizar la misma estructura de cobertura sobre cualquier motor de inferencia. Lo que se pretende es que se puedan llevar a cabo pruebas en el futuro que involucren a otros motores de reglas. También se ha pretendido que el nivel de dependencia con la fuente de información sea lo más bajo posible. Es muy importante puesto que en diferentes entornos se pueden tener diferentes necesidades respecto a donde deben estar almacenados los hechos. Por todo ello se ha llevado a cabo un diseño orientado a interfaces, delegando a las implementaciones de las interfaces las dependencias concretas con el motor de inferencia y con la base de hechos.

Capítulo 5

Validación

Una vez descrito el trabajo realizado conviene comprobar si los objetivos que se fijaron se han alcanzando o en qué medida se han concretado.

Esencialmente los objetivos a cumplir eran comprobar la viabilidad y si las funcionalidades que nos aporta un BRMS (sistema gestor de reglas de negocio) mejoran el proceso de desarrollo de las reglas. Para realizar la evaluación se ha llevado a cabo el desarrollo del módulo de calidad del motor de arbitraje del simulador SIMBA (Simulation in Business Administration).

Viabilidad.

Para comprobar la viabilidad del sistema se ha trabajado a partir de un documento inicial dónde se concretaba cada uno de los cálculos y su resultado. Por tanto el objetivo era construir a partir de dicho documento un conjunto de reglas que llevase a cabo los cálculos requeridos.

Uno de los pasos más importantes antes de elaborar el conjunto de reglas consiste en estudiar el lenguaje de definición de reglas que soporta drools y las limitaciones de este lenguaje impuestas por el editor de reglas de drools-guvnor. Una vez realizado este análisis se identificaron los elementos del conjunto de operaciones que realiza el módulo de simulación en lo referente a calidad que podrían ser más complicadas de escribir con drools y se determinó si realmente era posible construirlas en reglas. El resultado del estudio de la documentación fue positivo.

Incrementalmente se crearon los conjuntos de reglas, evaluando paso a paso si los resultados que se estaban obteniendo coincidían con los resultados esperados. Como conjunto de datos de test se tomaron los resultados del conjunto de fórmulas que se facilitaron inicialmente.

Tras implementar completamente el módulo de calidad podemos determinar que es **viable** construir un conjunto de reglas en drools, que tenga el mismo comportamiento que

el módulo de arbitraje del emulador de SIMBA, utilizando las herramientas que ofrece la plataforma.

Cuando se definieron los objetivos se determinaron una serie de requisitos cuyo cumplimiento sería deseable. Habría que resaltar en esta evaluación que se han cumplido casi todos. Por un lado el conjunto de reglas originado necesita de unos conocimientos de drools, y además, a la hora de determinar los tipos de los atributos del modelo, puede que no se haya seguido completamente las recomendaciones de Java para el tipado de elementos, en concreto la utilización de la interfaz Number para definir el tipado de algunos números del modelo. En concreto la recomendación es utilizar el tipo de datos más concreto posible en una jerarquía de objetos. Sin embargo estas concesiones son menores y se puede considerar que el objetivo se ha cumplido perfectamente.

Funcionalidades como BRMS.

El principal objetivo era evaluar las posibilidades que ofrece drools-guvnor como BRMS, en lo referente al problema concreto que ofrece el módulo de simulación de SIMBA. Para cada requisito que se planteó como importante se van a describir las principales ventajas que aporta al desarrollo de reglas frente a afrontar el desarrollo sin este tipo de herramientas. Las funcionalidades que se han evaluado son:

- **Edición de reglas**.
- **Definición de la ontología**.
- **Gestión de versiones**.
- **Gestión de errores**.
- **Seguridad**.
- **Integración**.

Edición de reglas

Para realizar el estudio se ha empleado el editor guiado de reglas. Este requisito es muy importante, ya que facilita tanto la edición como la modificación de las reglas, que son el eje central de un motor de reglas. De todos los editores de los que disponer drools-guvnor el que se está evaluando es el editor guiado. Este editor nos permite crear reglas a través de unos formularios en los que se va seleccionando los antecedentes las condiciones de los antecedentes, los consecuentes, el tipo de consecuente y demás elementos para definir una regla en drools.

Las principales **ventajas** son:

- **Facilita sintaxis**.
  Drools-guvnor permite realizar las operaciones más comunes que el lenguaje considera válidas, comparaciones, selección de hechos, modificaciones sobre la memoria de trabajo, etc... seleccionándolas mediante combos de selección o

elementos de formulario. De esta forma no es necesario conocer cómo la sintaxis nos permite escribir estas estructuras, simplemente podemos definir una regla con estas operaciones de manera trivial. De esta manera se evitan fallos de sintaxis, porque realmente el usuario no escribe las estructuras que está empleando. Este modelo de escribir reglas en drools tiene sus limitaciones y algunas de las más importantes las describiremos más adelante.

- **Integración con el modelo**.
  Durante la edición de las reglas sólo se trabaja con los elementos definidos en la ontología, se puede seleccionar una clase, sobre esa clase escoger los atributos, y sobre esos atributos, realizar las operaciones apropiadas.

- **Copia de reglas**.
  Una herramienta especialmente útil que permite avanzar rápido en el desarrollo de muchos proyectos de reglas. Como se ha comentado existen conjuntos de reglas con similares estructuras, al permitirse realizar copias de éstas reglas se puede avanzar mucho más rápido.

- **Localización y clasificación de reglas**.
  Un elemento muy importante es localizar las reglas. Sin el uso de estas herramientas se suelen escribir todas las reglas en un único fichero, ésto tiene muchísimos inconvenientes. Respecto a la edición de reglas, poder localizar fácilmente las reglas y poderlas agrupar por categorías, facilita mucho el trabajo con conjuntos de reglas grandes.

- **Edición transparente**.
  Un aspecto muy importante es que el proceso de edición es independiente de la representación física de las reglas. Gracias a eso las personas que editan las reglas no necesitan entender cómo se deben presentar las reglas al motor de inferencia.

Además de las ventajas que acabamos de mencionar, el editor guiado para crear o modificar reglas, incluye otra serie de funcionalidades bastante interesantes, como pueden ser: permite ver el código fuente de las reglas que se están creando, permite verificar que la regla está correctamente escrita individualmente, permite añadir directamente código fuente en drl, permite configurar el conjunto de elementos de la ontología con los que se desea trabajar.

Si en vez de emplear drools-guvnor empleásemos CLIPS, se perderían todas la ventajas que se han comentado. Tradicionalmente el trabajo con CLIPS se realiza sobre un único fichero donde se construyen todas las reglas. La máxima ayuda que ofrecen los editores es el resalto de las palabras claves del lenguaje. No existen herramientas que comprueben los errores que se cometan al escribir las reglas al margen del propio entorno de CLIPS, que no incluye ninguna utilidad de edición de reglas.

Aunque es una herramienta bastante útil, existen ciertas funcionalidades que se presentan mejorables o que simplemente no se han tenido en cuenta. Los aspectos que necesitan mejorar para poder ser más útiles son:

- **Funciones**.

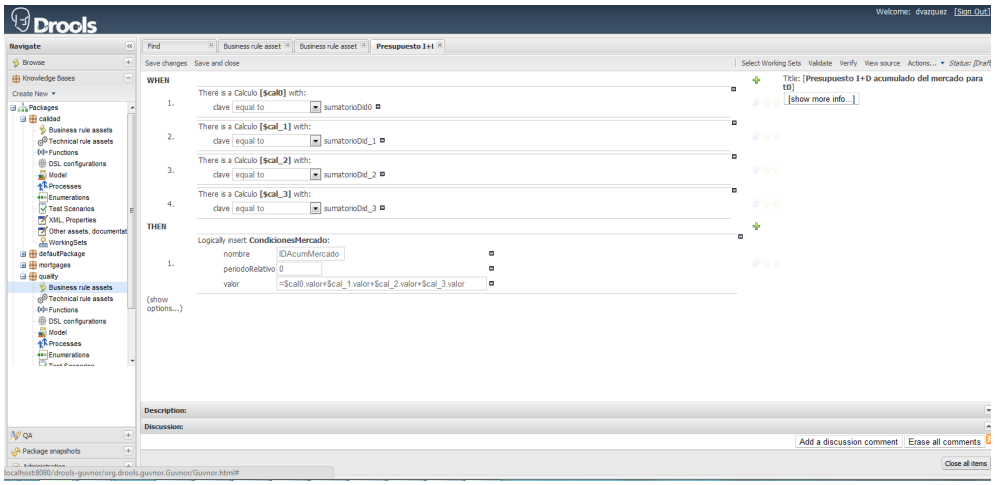


Figura 5.1: Pantalla de edición de reglas.

Aunque existe un editor de funciones, éste no es un editor guiado y requiere que se desarrollen por personas con conocimientos en la sintaxis de drools. Aunque ésto puede ser un inconveniente sin más, puesto que las funciones sólo se desarrollan una vez, es un elemento que probablemente requiera menos mantenimiento, al menos no en todos los problemas pero sí en el problema que se está resolviendo. Sin embargo el mayor problema que presenta drools-guvnr a este respecto es que no ”permite”que se utilicen las funciones que se definen en el editor guiado.

- **Necesario dominar ciertos elementos del lenguaje**.
  Dado el estado actual de drools-guvnor es imprescindible tener conocimientos del lenguaje de definición de reglas para resolver satisfactoriamente el problema propuesto.

- **Tipado débil**.
  El editor de reglas no realiza comprobaciones de tipos cuando se realizan asignaciones, incluso en algunas comparaciones con campo libre. Más adelante entraremos más en profundidad, pero incluso aunque estas asignaciones/comparaciones provoquen errores semánticos son permitidas al utilizar el editor guiado para crear las reglas. No ocurre con todos los casos y si se hace una correcta utilización de operadores de comparación en función de la parte izquierda de la operación, se podría paliar con una correcta detección de errores. Es un error bastante grave porque el motor de inferencia emplea un tipado fuerte.

- **Editores matemáticos**.
  No existen editores para fórmulas matemáticas, siendo necesario para construir una fórmula matemática escribirla empleando EL.

- **Refactorización**.
  Los editores modernos que se emplean en el desarrollo de sistemas in-

formáticos, suelen tener integradas herramientas que facilitan la realización de cambios de nombre en elementos donde cambiar un nombre puede originar un número elevado de errores. Por ejemplo si desarrollamos una aplicación en Java cambiar el nombre de una clase provoca que todas las referencias al nombre anterior de la clase se conviertan en errores de compilación. Al proceso de modificar estos nombres sensibles y a la vez corregir los errores derivados del cambio, se denomina refactorización. Drools-guvnor no incluye ningún elemento que ayude a la refactorización. Respecto del conjunto de reglas no es una funcionalidad imprescindible, aunque sí deseable.

- **Colecciones**.
  En la versión actual del editor no se pueden aprovechar las ventajas del editor guiado sobre un atributo que sea una colección dinámica de elementos.
- **Bugs**.
  A lo largo del uso de la herramienta se han encontrado diversos bugs que ha n dificultado el trabajo llevado a cabo con el editor.

La mayoría de las dificultades descritas afectan al problema concreto que se propone resolver (construir la parte de calidad del módulo de arbitraje). Es el dominio del problema el que nos exige utilizar ciertos elementos del lenguaje que no son muy empleados (como el operador from) y que no son completamente soportados por el editor guiado. Si este operador fuese mejor soportado, los problemas relativos a las funciones o al conocimiento necesario del lenguaje, serían más leves. Los problemas relativos al tipado débil y a la necesidad de editores de operaciones matemáticas, también están derivados del problema del simulador SIMBA. La necesidad del simulador de realizar un gran número de operaciones matemáticas complejas hacen resaltar lo útil que sería un editor de operaciones matemáticas. Quizás el inconveniente más importante para evaluar la viabilidad que presenta el editor guiado sean los bugs que tiene.

Aunque no directamente relacionado con la edición de reglas un aspecto importante a mejorar es la búsqueda de reglas ya que estas no se indexan en la versión actual y sí sería un aspecto muy interesante que ésto fuese así.

Respecto a los otros editores que dispone drools-guvnor, el uso de flujos de reglas es una decisión de diseño del problema que tras un breve estudio se descartó, las matrices no servirían para resolverlo y la utilización del lenguaje dsl en vez de drl no nos serviría para construir el simulador. Por supuesto se descarta la opción de construir las reglas directamente en drl, puesto que lo que se persigue es que el conocimiento de drools sea el menor posible para las personas que construyen las reglas. Por no ser necesarios para resolver el problema, no se han evaluado cómo funcionan estos editores de reglas.

Definición de la ontología

Un paso muy importante dentro de un motor de reglas es la definición de la ontología. Por tanto es necesario evaluar las herramientas de las que dispone drools-guvnor para definirla.

Anteriormente ya hemos comentado que es especialmente interesante la perfecta integración entre la ontología cuando se define y el editor de reglas.
Las principales **ventajas** son:

- **Proceso guiado**.
  Similar a como sucede con las reglas existe un proceso guiado que sirve de apoyo para la construcción de la ontología. El proceso es bastante sencillo, consiste simplemente en una serie de formularios para escribir el nombre de una entidad, escribir los nombres de sus atributos y los tipos de los mismos. El propio editor tiene sugerencias para los tipos de datos más comunes. Si fuese necesario utilizar algún tipo de dato que no está definido por el editor, incluye un mecanismo para poder escribir el tipo de dato que deseamos utilizar, aunque este debe ser un tipo soportado por la api de Java.

- **No requiere conocimientos**.
  El proceso guiado es suficientemente sencillo como para no requerir conocimientos de drools. Simplemente es necesario definir el nombre de la entidad, el nombre de cada atributo y el tipo de cada atributo. El único elemento que podría requerir un conocimiento del lenguaje son los tipo de datos, sin embargo las sugerencias del editor cubren las necesidades más comunes, incluso sugiere utilizar como tipo de dato las entidades ya creadas con anterioridad.

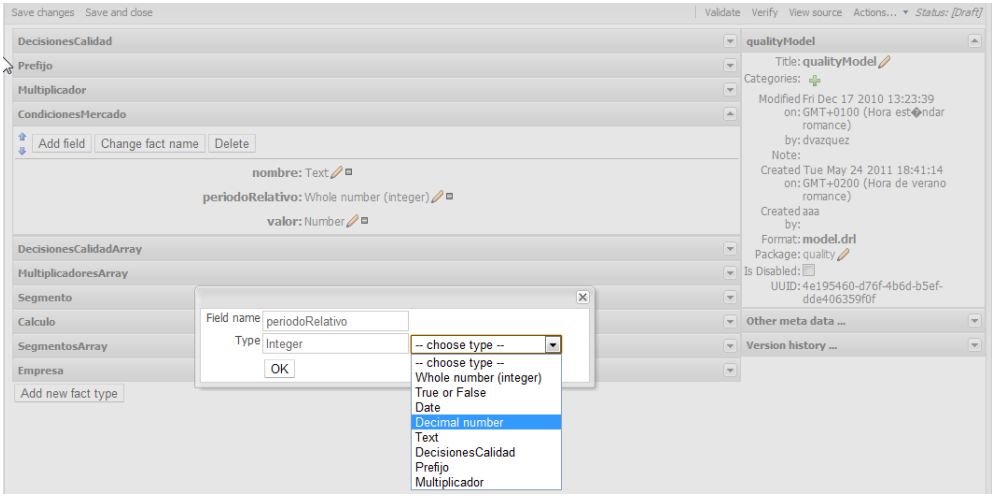


Figura 5.2: Pantalla de edición de la ontología.

Definir entidades en drools es bastante sencillo, pero requiere conocer el lenguaje de definición de las reglas. Emplear drools-guvnor nos previene de conocer cualquier entresijo del lenguaje. Respecto a CLIPS, esta ventaja es todavía más importante, porque es bastante más complejo definir la ontología con CLIPS que con drools.

Los principales **inconvenientes** son:

- **Refactorización**.
  La falta de esta funcionalidad se hace especialmente grave en la edición del

modelo. Si necesitamos modificar el modelo, el procedimiento actual consiste en realizar la modificación y compilar las reglas y corregir los errores que provoque la modificación que hemos realizado en el modelo. No existe ninguna herramienta que nos avise de qué reglas se van a ver afectadas a priori o que propague los cambios que realicemos.

- **Bugs**.
  Existen algunos bugs que dificultan el uso de la herramienta.

Gestión de versiones

Un gestor de versiones es una herramienta imprescindible para llevar a buen puerto un proyecto de desarrollo, y por supuesto para llevar a cabo el desarrollo del conjunto de reglas. Drools-govnur permite que las versiones de las reglas sean gestionadas tanto como reglas individuales, como gestionar un paquete conjunto de reglas para ser empleadas en un motor de inferencia. Sin hacer menoscabo de la importancia de gestionar las versiones de un paquete de reglas, es más interesante la funcionalidad de gestionar individualmente las reglas, porque es la más novedosa respecto de los BRMS.

La principal característica es que drools-govnur maneja la gestión de versiones de las reglas mediante un repositorio. Los repositorios de contenidos suelen tener muchas funcionalidades comunes a los gestores de versiones, como son el almacenamiento de cada versión; añadir meta-información clave como qué usuario hizo los cambios, en qué fecha, y unos comentarios sobre el cambio; permitir recuperar versiones anteriores; entre otras.

Sobre un paquete de reglas, drools-guvnor nos permite etiquetarlo para poder distinguir entre diferentes versiones del mismo conjunto de reglas. El motor de inferencia necesita un conjunto de reglas y es muy importante que estos conjuntos estén versionados.

Además de lo comentado algunas de las principales **ventajas** son:

- **Cumplimiento de estándares**.
  El repositorio que utiliza drools-guvnor cumple con los estándares de la Java Enterprise Edition (la especifiación jsr-170), con lo que se pueden modificar los parámetros del repositorio muy fácilmente siguiendo las indicaciones de la documentación ofrecida por el estándar.
- **Importación/Exportación**.
  Incluye la posibilidad de importar/exportar los datos de las reglas, facilitando el trabajo con varias instancias de drools-govnur y/o las migraciones de datos entre diferentes versiones.

Los aspectos que serían mejorables, son:

- **Configuración/instalación**.
  Cambiar el soporte sobre el que se almacena las reglas o la forma en la que se hace implican modificar los ficheros de configuración de la aplicación, y arrancar la aplicación. Es más una configuración de instalación que un elemento

Figura 5.3: Seleccionar la versión de una regla.

que podamos configurar en nuestra aplicación. Un cambio en el repositorio una vez se ha empezado a trabajar, supondría perder el trabajo que ya se ha realizado (podríamos sin ningún problema importar la información de las reglas, si hubiésemos realizado una exportación previamente). Generalmente la decisión sobre cómo va a funcionar el repositorio se debería tomar antes de la instalación. Con la configuración por defecto el repositorio cubre perfectamente las necesidades básicas.

- **Salvar sin almacenar**.
  No se pueden salvar cambios de una regla sin que se agregue una nueva versión de regla al repositorio y a la última versión del empaquetado de reglas.
- **Gestión de versiones paralelas (ramas)**.
  No incluye mecanismos que nos ayude a gestionar una rama sobre un paquete de reglas. Lo que es una grave deficiencia en un gestor de versiones.
- **Falta de funcionalidades**.
  Aunque las funcionalidades más importantes de un gestor de versiones están perfectamente implementadas, todavía faltan algunas por implementar que mejorarían la herramienta. En especial, falta alguna forma de comparar el código fuente de una regla con sus versiones anteriores.

Gestión de errores

Un aspecto muy importante es la detección de errores durante la edición, así como las ayudas que pueda ofrecer la herramienta para resolver dichos errores. Las principales **ventajas** son:

- **Compilación individualizada**. Drools-guvnor permite realizar una compilación individualizada del código de la regla gracias a lo cual nos permite evaluar directamente si la regla contiene errores o no.
- **Montaje del paquete de reglas**. Cuando se desea publicar un conjuntos de

reglas para que un motor externo acceda a ellas, es obligatorio que dicho paquete se compile completamente. Gracias a ésto se puede detectar si existe algún error en el conjunto de reglas y además a qué reglas afecta.

Los principales **inconvenientes** son:

- **No parametrización del compilador**.
  No se permite introducir parámetros que afectan a la compilación de las reglas, lo que puede forzar a acceder a las reglas directamente en código fuente o incluso dar errores cuando se introducen al motor.
- **Mala traducción de errores**.
  Los errores encontrados no suelen ser fácilmente entendibles, ya que hacen referencia al código fuente de la regla y desde la perspectiva del editor guiado de reglas puede resultar difícil deducir qué quiere decir el error.

Uno de los problemas principales que existe con la gestión de errores es que no es capaz de detectar todos los posibles errores que sí detecta el motor cuando va a ejecutar las reglas. Por ejemplo, ya se ha hecho referencia al problema tipado débil. Incluye problemas derivados de la posible parametrización del compilador en el entorno donde se va a emplear, incluso problemas derivados

Gestión de usuarios

Una funcionalidad bastante importante es la gestión de usuarios. No hay que olvidar que drools-guvnor es una herramienta distribuida desde la que varios usuarios pueden colaborar para construir las reglas de un sistema basado en reglas. De hecho esta colaboración es muy común y por tanto es necesario diagnosticar cuáles son las virtudes y defectos de la gestión de usuarios.

En cualquier aplicativo distribuido no sólo es importante la gestión de usuarios, sino también la gestión de privilegios. Por ejemplo un usuario que puede modificar reglas, no tiene por qué encargarse de realizar exportaciones de datos. Los privilegios son también un aspecto muy importante de drools-guvnor.

Las principales **ventajas** son:

- **Seguimiento de estándares**.
  El seguimiento de estándares en los aspectos de seguridad que realiza drools-guvnor, permite conocer fácilmente cuáles son los parámetros de seguridad que se pueden modificar y cómo se pueden modificar. De esta forma se facilita la integración a sistemas de gestión de usuarios que existiese en la organización donde se va a implantar drools-guvnr.
- **Configuración de privilegios**.
  La configuración de los privilegios que se puede realizar desde la interfaz es muy intuitiva y sencilla. Los privilegios se basan en dos aspectos, las operaciones que se pueden realizar (administrador, analista, analista de sólo lectura,administrador de proyectos, desarrollador y consultor de proyectos) en los

proyectos y/o categorías sobre los que se pueden realizar. Las convenciones que se pueden dar parecen suficientemente potentes como para cubrir todas las posibles necesidades que surjan al respecto.

- **Comunicación**.
  Se incluyen mecanismos para que los usuarios puedan recibir notificaciones de otros usuarios y así facilitar el trabajo realizado de manera concurrente.

La configuración por defecto de drools-guvnor es que todo usuario que haga login queda inscrito como usuario del aplicativo. Aunque esto no es la situación ideal, no es una mala solución, porque cada organización tiene su propia gestión de usuarios y en general prefieren que los usuarios sean los mismos entre las diferentes aplicaciones que emplean.

Ámbito	Rol	Descripción
Aplicación	administrador	No sólo tiene completo control sobre todos los paquetes de reglas, sino también se encarga de gestionar los usuarios de la aplicación y todos los demás elementos de la misma.
Paquete	administrador	Puede gestionar completamente un paquete de reglas, pero no puede gestionar ningún aspecto de la plataforma.
Paquete	desarrollador	Puede crear cualquier item de un paquete de reglas (reglas, modificar la ontología...), sin embargo .
Paquete	sólo lectura	Puede consultar la información del paquete pero no puede modificar nada.
Categoría	analista	Sólo puede consultar unas categorías y las reglas asociadas a dichas categorías, se pueden combinar con los privilegios relacionados con los paquetes.
Categoría	analista de sólo lectura	Es similar a los permisos de analyst, pero no se puede realizar ningún tipo de modificación.

El principal inconveniente a destacar es que de manera similar a como ocurría con el repositorio, los cambios de configuración sobre la gestión de usuarios requieren que se modifique la configuración durante la instalación del sistema. Pudiendo llegar a ser necesario, llevar a cabo un pequeño desarrollo para poder realizar una integración perfecta con el sistema de autenticación que ya se desease emplear.

Otro inconveniente es la falta de mecanismos que permitan proteger una regla mientras un usuario está trabajando con ella. Dos usuarios pueden estar trabajando en una regla a la vez sin que el sistema advierta a ninguno de los dos o se establezcan mecanismos para

dinamizar este problema.

Integración

Sin mecanismos sencillos de integración un BRMS sería una estupenda herramienta de desarrollo, pero quedaría una parte bastante importante por cubrir respecto a ser empleado en un entorno real. A ese respecto drools-guvnor publica las diferentes versiones de los diferentes paquetes de reglas a través de una url. Respecto al diseño de la arquitectura ésto tiene una importante implicación y es que el motor de inferencia tiene que poder acceder al lugar donde se encuentra instalado drools-guvnor, lo cual no es siempre posible.

Las principales **ventajas** son:

- **Distribuido en diferentes formatos**.
  Drools-guvnor está enfocado a trabajar junto con el motor de drools. Esto implica que el motor puede cargar perfectamente un conjunto de reglas desde un recurso que es accesible vía http (que es justo el mecanismo de publicación de paquetes de reglas de drools-guvnor). Soporta dos formatos para exportar las reglas, uno en texto plano, y otro en un formato serializado, el motor soporta perfectamente estos dos formatos. La gran ventaja es que sea cual sea la forma en que se construyan las reglas al final siempre se transmiten de la misma forma.
- **Distinción de versiones**.
  Respecto a las versiones realizadas sobre el empaquetado de reglas, todas son accesibles, incluyendo el último estado de las reglas que pertenecen al paquete y que no han sido etiquetadas.
- **Seguro**.
  El acceso a la url del paquete de reglas se encuentra protegido por un sistema de usuario y contraseña, de esta forma sólo podría acceder a ésta url un usuario registrado y con los permisos pertinentes.

Los principales **inconvenientes** son:

- **Doble compilación**.
  Sólo se publican conjuntos de reglas que han sido compilados en el BRMS. Sin embargo lo que debería ser una garantía de que el motor puede cargar todas las reglas de manera segura, no es cierto, el compilador de drools-guvnor no evalúa ciertos problemas relacionados con el tipado de los elementos, que posteriormente al cargar las reglas provocan determinados errores.

Otros aspectos

Para que un BRMS sea considerado como tal, debe de tener su propio motor de inferencia integrado. Ésto es necesario para cumplir tres tareas fundamentales, realizar pruebas, comprobar que las reglas son válidas y ejecutar el motor. Drools-guvnor tiene un motor integrado, sin embargo presenta diversas lagunas en los tres puntos.

Drools-guvnor tiene su propio entorno de pruebas, sin embargo, debido a que la ontología que se ha empleado para resolver el problema del módulo de arbitraje de SIMBA es muy compleja, no es compatible con dicho entorno de pruebas. El problema radica en que el entorno de pruebas no puede cargar hechos correctamente si los elementos de la ontología tienen relaciones entre sí.

Drools-guvnor no permite solamente el motor de interferencia, pero sí que tiene uno integrado, que le permite entre otras cosas poder ejecutar test unitarios.

Otro aspecto negativo de la plataforma es la falta de documentación. Las funcionalidades no está suficientemente bien documentadas. También en el mismo orden pero con distinto impacto, tampoco el código fuente está suficientemente documentado.

Diferencias entre drools CLIPS.

No se pretende establecer las diferencias entre ambas herramientas, sino especificar las principales diferencias entre las soluciones que ambas han alcanzado para resolver este problema. Hay que tener en cuenta que ambas soluciones se basan en elaborar un sistema de reglas.

Las reglas que se utilizan en la solución elaborada con drools, son más legibles que las calculadas con CLIPS. Ésto facilitaría una mayor implicación por parte de las personas que mejor conocen el dominio en la parte técnica del proyecto.

No existe en la solución de drools ningún tipo de asunción de valores. En la solución de CLIPS, se presuponen determinados valores que en el momento en el que se definió el problema son constantes, por ejemplo los segmentos a los que pueden pertenecer las empresas. Este aspecto no invalida la solución de CLIPS puesto que los valores se han definido como constantes, sin embargo añaden una dependencia en las reglas con los datos que puede originar problemas si fuese necesario modificar o añadir nuevos valores.

Otro aspecto es que las reglas de CLIPS han sido diseñadas para que el factor de prioridad sea muy importante, introduciendo la sensación de que se ejecutan secuencialmente. Las reglas de drools establecen dependencias establecidas únicamente en función de sus antecedentes. Si dos reglas está en el mismo conjunto de colisión, es indiferente para el resultado final cuál de ellas se ejecuta antes. El gran problema de enumerar todas las reglas en función de su prioridad (que es lo que ocurre en CLIPS) es que introducir una nueva regla puede originar un efecto cascada sobre las demás reglas de imprevisibles consecuencias.

En la solución de CLIPS se ha puesto más empeño en conseguir un sistema que tenga un gran rendimiento, primando este factor sobre otros. En la solución de drools se ha intentado construir un sistema más fácilmente mantenible y más sencillo de comprender.

Además de las facilidades que da el motor de reglas de drools para resolver este proble-

ma, frente al de CLIPS, es importante recordar que drools-guvnor nos incluye una serie de funcionalidades que facilitan en gran medida el trabajo y que no existe de manera integrada para desarrollar reglas en entorno CLIPS, algunos de los puntos más importantes son:

- **Edición de reglas**: No existen editores de reglas para CLIPS, más allá de editores de texto o algún resaltador de sintaxis. Además el lenguaje de definición es ostensiblemente más complejo en CLIPS, lo que dificulta alcanzar soluciones de editores particulares para un problema concreto.
- **Gestión de versiones**: Para integrar un desarrollo en CLIPS en un gestor de versiones es necesario emplear un gestor que se emplee en un desarrollo de software normal. Esta no es la mejor solución, ya que las reglas, en general, se escriben en pocas líneas y los sistemas suelen tener muchas reglas. Mantener un fichero de texto por cada regla sería muy costoso, si el número de reglas fuese relativamente alto sería prácticamente imposible manejarlo.
- **Integración base de reglas**: No existen mecanismos que nos permitan integrar una base de reglas en CLIPS más allá de cargar un fichero.

Estos puntos pueden parecer fácilmente subsanables, sin embargo debemos tener en cuenta que el objetivo principal de un sistema basado en reglas es que un usuarios que no posea conocimientos de programación sea capaz de desarrollar reglas. Actualmente no existen herramientas orientadas a estos usuario para CLIPS.

CLIPS es un una gran herramienta que ha sido empleada en multitud de problemas con éxito. Sin embargo drools es una tecnología mucho más joven que todavía está en crecimiento.

Capítulo 6

Gestion del Proyecto

A continuación se van a detallar los aspectos referentes a la gestión del proyecto tales como los modelos seguidos, tiempo y recursos utilizados.

6.1. Metodología

Se ha empleado una metodología orientada la ingeniería del software, porque lo que se pretende es sustituir la implementación del simulador por otra que se ajuste mejor a los problemas ya referidos.

Plantearse utilizar ingeniería basada en el conocimiento hubiese sido innecesario, aunque SIMBA sea un sistema basado en el conocimiento. Todos los problemas que se afrontan con la ingeniería del conocimiento ya están resueltos, ya estaba identificado el problema, ya estaba definido el conocimiento necesario para resolver el problema, etc...

Para tratar de optimizar las tareas necesarias para completar el proyecto, se ha decidido llevar a cabo un desarrollo basado en *espiral* ([14]). El objetivo al utilizar esta metodología consiste en completar ciclos de desarrollo que nos permitan tener partes funcionales acabadas e ir ampliando progresivamente el estado.

Dada los problemas en la disponibilidad de recursos con la que nos hemos encontrado, el riesgo existente en no poder definir correctamente parte del sistema o las relaciones tan intrínsecas entre las reglas (unas construyen hechos necesarios para otras), son las razones por las que principalmente han llevado a tomar la decisión de gestionar las tareas del proyecto con este ciclo de vida.

El desarrollo en espiral es un modelo de ciclo de vida del software definido por primera vez por Barry Boehm en 1988, utilizado generalmente en la Ingeniería de software. Las actividades de este modelo se conforman en una espiral, en la que cada bucle o iteración representa un conjunto de actividades. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior.

Por tanto en cada iteracción o bucle se llevan a cabo una serie de tareas que nos acercan a la solución del problema. En éste caso de aplicación se ha perseguido que cada iteracción de como resultado un elemento tangible del proyecto.

En cada iteración está dividida en cuatro fases:

- **Determinar o fijar objetivos**. Determina los objetivos a cumplir en éste bucle. Fija cuáles se desean que sean los resultados al terminar el bucle.
- **Análisis del riesgo**. Se analiza si los objetios son pausibles o si la solución propuesta se aleja del objetivo final. Al finalizar ésta fase es importante determinar si es viable el trabajo o si es necesario abandonarlo.
- **Desarrollar, verificar y validar**. Se llevan a cabo las tareas necesarias para cumplir con los objetivos propuestos. Pueden tener asociados a su vez otro ciclo de vida. Típicamente se van construyendo prototipos de la aplicación que se está desrrollando.
- **Planificar**. Planificación de la siguiente interación en función de los resultados que se han obtenido.

Las principales ventajas que incorpora utilizar éste ciclo de vida son:

- Incorpora muchas de las ventajas de los otros ciclos de vida.
- Conjuga la naturaleza iterativa de los prototipos con los aspectos controlados y sistemáticos del modelo clásico.
- Proporciona el potencial para el desarrollo rápido de versiones incrementales.
- Puede adaptarse y aplicarse a lo largo de la vida del software.
- Es un enfoque realista del desarrollo del software.
- Permite aplicar el enfoque de construcción de prototipos en cualquier momento para reducir riesgos.
- Reduce los riesgos antes de que se conviertan en problemáticos.
- Controla muy bien los riesgos y mientras más iteraciones se realicen, menos riesgos habrá.
- Monitoriza y controla la calidad continuamente.

Algunas de sus desventajas son:

- Puede resultar difícil de entender el enfoque evolutivo que persigue el ciclo.
- Puede suponer una carga de trabajo adicional, no presente en otros ciclos de vida.
- Requiere una considerable habilidad para la evaluación y resolución del riesgo, y se basa en esta habilidad para el éxito.

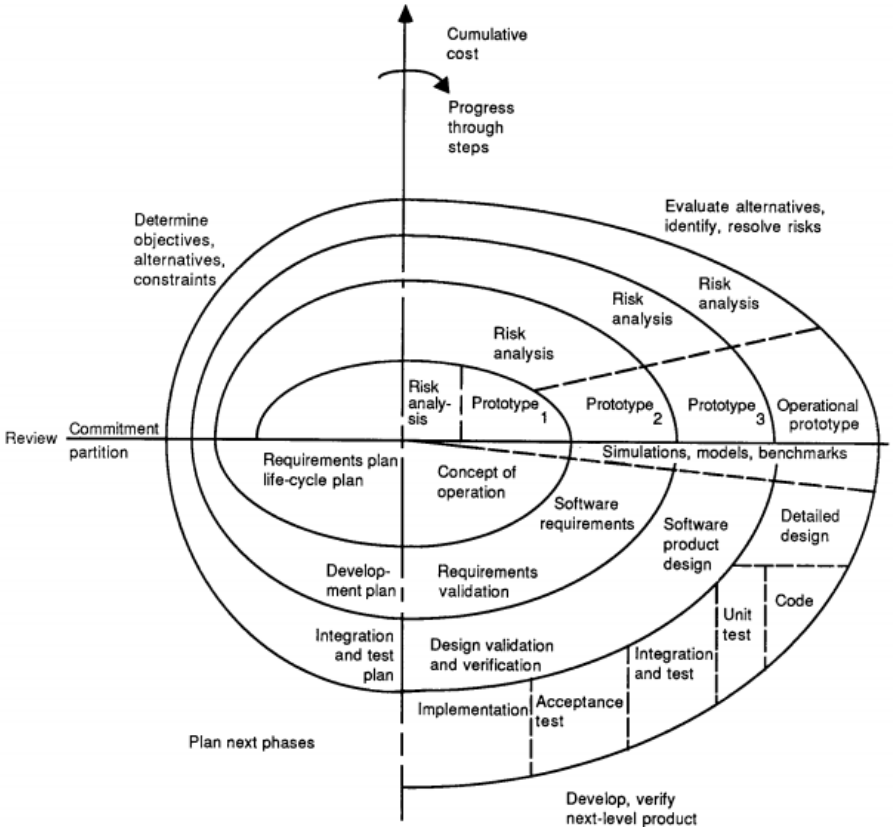


Figura 6.1: Diagrama de cómo deben de ser las diferentes interacciones del modelo en espiral *espiral*.

Hay ciertos elementos de éste ciclo de vida que no se han respetado. En cada interacción se construye un prototipo de la aplicación que se va ampliando incrementalmente hasta cubrir completamente la aplicación. Estrictamente hablando no se ha segido ésta filosofía. Los ciclos han sido bastante pequeños (en cuanto a objetivos a cumplir se refiere), pero acorde a la envergadura del proyecto y a la disponibilidad de recursos. Elaborar prototipos del sistema de reglas hubiese supuesto modificar en diferentes iteracciones las mismas reglas, lo que hubiese supuesto añadir un elemento de inestabilidad sobre las pruebas del sistema, en contraposición ha resultado más sencillo un desarrollo modular del conjunto de reglas.

6.2. Ciclo de vida

Para llevar a cabo el completo desarrollo del proyecto se han llevado a cabo siete interacciones. Los objetivos de cada interacción han sido:

- **Interacción 0**: Estudio inicial.

- **Interacción 1**: Ontología, Reglas de Cálculo.
- **Interacción 2**: Cobertura sobre drools-expert 1.0, reglas de Condiciones de Mercado.
- **Interacción 3**: Revisión Ontología Inicial, revisión Reglas de Cálculo, revisión de reglas de Condiciones de Mercado.
- **Interacción 4**: Cobertura sobre drools-expert 1.1.
- **Interacción 5**: Reglas de Selección de multiplicadores.
- **Interacción 6**: Revisión final de la Ontología, Regla de Cálculo de calidad.

Los objetios de cada interacción se marcaron en función de la disponiblidad de recursos (poder cerrar un interacción en un tiempo razonable de tiempo y que no hubiese interacciones que se viesen paradas) y tratando que el resultado de cada interacción fuese un elemento completo.

Interacción 0: Estudio inicial.

Ésta es una interacción un poco especial dentro del modelo en espiral, en la que se realizan una serie de tareas que suelen afectar a todas las interacciones. Éstas tareas suelen ser la planificación la adquisición de requisitos, el análisis de la arquitectura...

En el caso del siguiente proyecto, se realizó un estudio de la documentación de drools, una primera aproximación sobre cómo se representarían las fórmulas matemáticas utilizando reglas basadas en drl y un estudio de las opciones que tenía drools-guvnor. Además se llevó a cabo una planificación.

Este primer análisis se llevó a cabo en 104 horas.

Interacción 1: Ontología, Reglas de Cálculo.

Los objetivos que se marcaron fueron elaborar la ontología y construir las reglas de Cálculo.

El análisis de riesgo se centró en la flexisibilidad a la hora de definir la ontología y en el uso de las herramientas de drools-guvnor para definir la ontología, y construir reglas (en especial de manejar el operador from).

En el análisis de riesgos se determinó que no era posible utilizar drools-guvnor como plataforma de pruebas.

Para la siguiente interacción se fijó como necesario construir una plataforma para poder realizar pruebas sobre las reglas.

Esta primera interacción se llevó a cabo en 98 horas.

Interacción 2: Cobertura sobre drools-expert 1.0, reglas de Condiciones de Mercado.

Los objetivos que se marcaron fueron elaborar una primera versión de una sofware que nos permitiese realizar pruebas al margen de drools-guvnor y construir el siguiente conjunto de reglas las reglas referentes a las condiciones de mercado.

Durante el análisis se percibió que con la ontología actual no era posible representar todas las reglas de condiciones de mercado y que sería necesario añadir más reglas referente a los cálculos preeliminares de las que se había añadido.

Debido a los problemas encontrados no fue posible cumplir plenamente el objetivo de construir las reglas relacionadas con las condiciones de mercado. Por tanto para la siguiente interacción se trasladaron los problemas aquí encontrados.

Esta segunda interacción se llevó a cabo en 80 horas.

Interacción 3: Revisión Ontología Inicial, revisión Reglas de Cálculo, revisión de reglas de Condiciones de Mercado.

Ante los problemas encontrados anteriormente, durante este interacción se resolvieron. Se revisó la ontología inicial, para cubrir las necesidades de todas las reglas de las condiciones de mercado. Se corrigieron los problemas derivados de la falta de algunas reglas de cálculo y con todos éstos problemas resuletos se completó el conjunto de las reglas referentes a las coniciones de mercado.

En la evaluación de ésta interacción se comprobó que era necesario mejorar el software desarrollado para pruebas.

Esta tercera interacción se llevó a cabo en 70 horas.

Interacción 4: Cobertura sobre drools-expert 1.1.

En ésta interacción se buscó mejorar el software para realizar pruebas. Debido a que el siguiente conjunto de reglas era muy grande y complicado de probar no se incluyó en ésta interacción.

Esta cuarta interacción se llevó a cabo en 20 horas.

Interacción 5: Reglas de Selección de multiplicadores.

En ésta interacción se construyó el conjunto de reglas que se encargan de la selección de multiplicadores.

Esta quinta interacción se llevó a cabo en 100 horas.

Interacción 6: Revisión final de la Ontología, Regla de Cálculo de calidad.

Por último faltaba la construcción de la regla de rating de calidad.

Durante el análisis de ésta regla se detectaron ciertos errores sobre alguntos tipos de datos de la ontología. Se aprovechó esta interacción para resolverlos.

Esta sexta interacción se llevó a cabo en 56 horas.

6.3. Planificacion del proyecto

Tareas

Las tareas definidas se han organizado como se puede apreciar en la figura 6.2, y su correspondiente diagrama de Gantt (6.3).

	Nombre	Duración	Inicio	Terminado	Predecesores
1	**Interacción 0**	52 days?	**18/02/10 9:00**	**3/05/10 9:00**	
2	Estudio de la documentación	20 days?	18/02/10 9:00	18/03/10 9:00	
3	Definición del problema	15 days?	18/03/10 9:00	8/04/10 9:00	2
4	Estudio del módulo de calidad	10 days?	8/04/10 9:00	22/04/10 9:00	3
5	Definición de las tareas	7 days?	22/04/10 9:00	3/05/10 9:00	4
6	**Interacción 1**	49 days?	**3/05/10 9:00**	**9/07/10 9:00**	**1**
7	Análisis de las fórmulas	5 days?	3/05/10 9:00	10/05/10 9:00	
8	Estudio de drools-guvnor	15 days?	10/05/10 9:00	31/05/10 9:00	7
9	Definición del modelo de conocimiento	10 days?	31/05/10 9:00	14/06/10 9:00	8
10	Definición de las reglas de cálculo	15 days?	14/06/10 9:00	5/07/10 9:00	9
11	Pruebas de las reglas de cálculo	4 days?	5/07/10 9:00	9/07/10 9:00	10
12	**Interacción 2**	40 days?	**9/07/10 9:00**	**3/09/10 9:00**	**6**
13	Análisis de las fórmulas de condiciones de mercado	4 days?	9/07/10 9:00	15/07/10 9:00	
14	Definición de las reglas de condiciones de mercado 1	8 days?	15/07/10 9:00	27/07/10 9:00	13
15	Elaboración de cobertura sobre drools-expert	10 days?	27/07/10 9:00	10/08/10 9:00	14
16	Pruebas de la cobertura sobre drools-expert	3 days?	10/08/10 9:00	13/08/10 9:00	15
17	Pruebas de las reglas de condición de mercado 1	15 days?	13/08/10 9:00	3/09/10 9:00	16
18	**Interacción 3**	35 days?	**3/09/10 9:00**	**22/10/10 9:00**	**12**
19	Revisión del modelo de conocimiento	1 day?	3/09/10 9:00	6/09/10 9:00	
20	Revisión de las reglas de cálculo	14 days?	6/09/10 9:00	24/09/10 9:00	19
21	Definición de las reglas de condiciones de mercado 2	4 days?	24/09/10 9:00	30/09/10 9:00	20
22	Prueba de las reglas construidas	16 days?	30/09/10 9:00	22/10/10 9:00	21
23	**Interacción 4**	10 days?	**22/10/10 9:00**	**5/11/10 9:00**	**18**
24	Mejoras sobre la cobertura de drools-expert	5 days?	22/10/10 9:00	29/10/10 9:00	
25	Prueba de las reglas construidas	5 days?	29/10/10 9:00	5/11/10 9:00	24
26	**Interacción 5**	50 days?	**5/11/10 9:00**	**14/01/11 9:00**	**23**
27	Reglas de selección de multiplicadores	25 days?	5/11/10 9:00	10/12/10 9:00	
28	Prueba de reglas	25 days?	10/12/10 9:00	14/01/11 9:00	27
29	**Interacción 6**	28 days?	**14/01/11 9:00**	**23/02/11 9:00**	**26**
30	Regla de rating de calidad	8 days?	14/01/11 9:00	26/01/11 9:00	
31	Prueba de reglas construidas	20 days?	26/01/11 9:00	23/02/11 9:00	30

Figura 6.2: Planificación de las tareas que se han llevado a cabo.

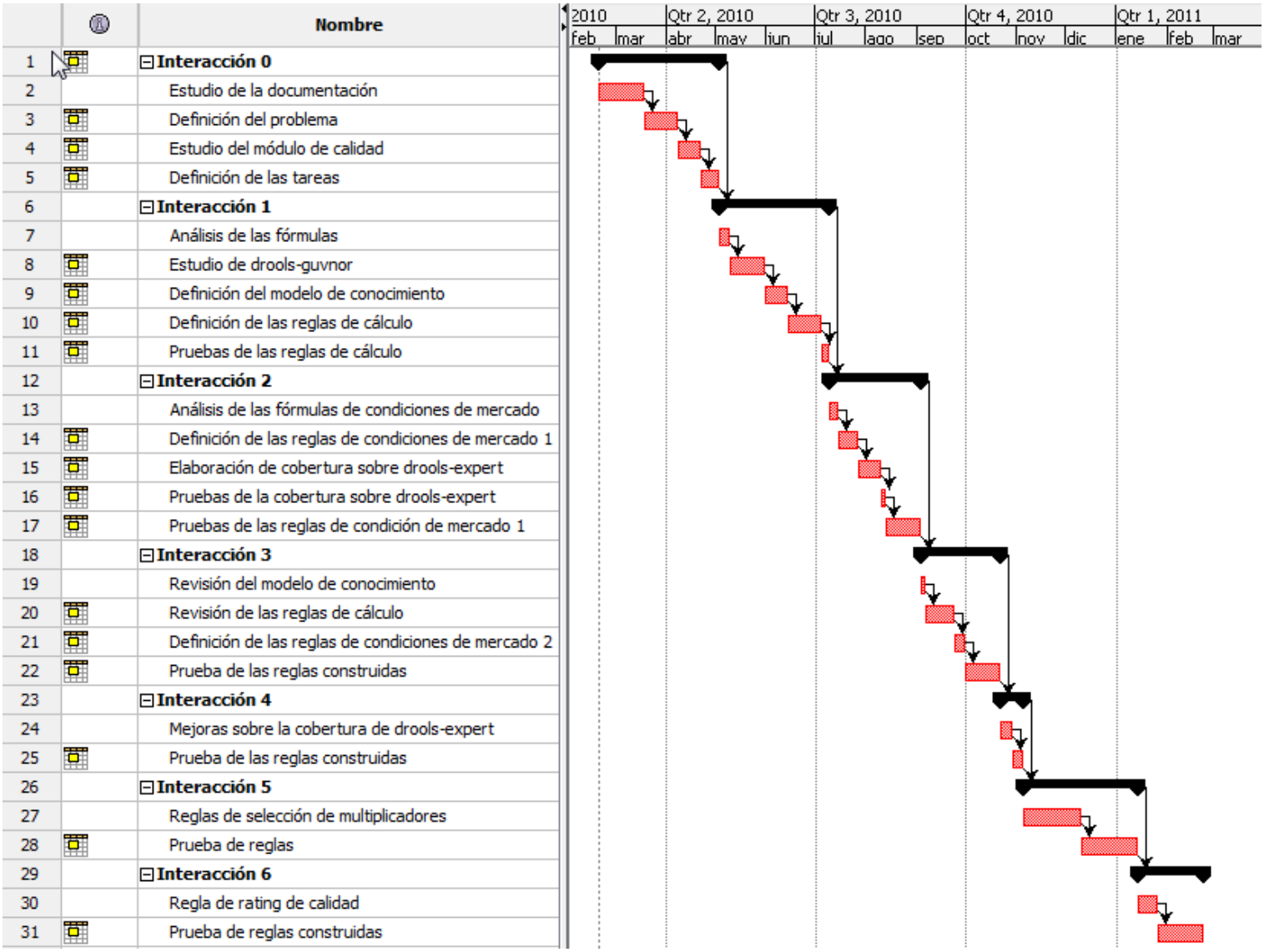


Figura 6.3: Diagrama de Gantt de las tareas realizadas.

6.4. Presupesto

A continuación se van a detallar los costes asociados a los recursos empleados para la elaboración del estudio. En principio no habrá consideraciones referentes a beneficios, ya que se considera que no es la finalidad de éste estudio.

Para la evaluación de los costes finales se tendrán en cuenta los costes proporcionalmente al tiempo que fueron utilizados durante el proyecto, respecto a la vida útil estimada de cada uno de ellos. Esto se debe a que, una vez terminado el mismo, los equipos y programas adquiridos podrán seguir siendo utilizados en futuros desarrollos.

Puesto que todos los recursos software que se han empleado en el estudio son gratuitos no se han incorporado al presupuesto, ya que sólo se detallarán los costes de aquellos componentes con precio mayor de 0€.

Recurso	**Precio**	**Vida útil**	**Uso**	**Coste para el proyecto**
Equipo informático	1.800€	24 meses	13 meses	975€.
Analista/Programador	30.000€	12 meses	13 meses	32.500€.
			TOTAL	33.475€.

Capítulo 7

Conclusiones y Trabajos Futuros

Una vez expuesto el estudio realizado faltan comentar las conclusiones que se han extraído del mismo, así como las líneas de trabajo que se recomiendan seguir en el futuro.

7.1. Conclusiones

Drools posibilita una solución de calidad al problema de construir el simulador de SIMBA en un sistema basado en reglas. Además la solución empleando drools es mejor que la solución basada en un lenguaje imperativo. La solución obtenida es más sencilla de evolucionar, permite localizar perfectamente las operaciones que se están realizando, está basada en un modelo de datos sencillo e intuitivo, pero sobre todo, si empleásemos esta solución, el ciclo de desarrollo del simulador sería independiente del desarrollo del resto de la aplicación, que es actualmente uno de los grandes problemas que tiene la evolución de SIMBA.

Comparando los conjuntos de reglas resultantes con el resultado de CLIPS, drools demuestra ser una mejor opción para éste problema. El conjunto de reglas resulta mucho más legible, más sencillo de interpretar y más fácil de evolucionar. Esta reflexión es únicamente respecto al problema que estamos tratando y no debe ser extrapolable a todos los casos. La ventaja de drools reside en su preparación para determinadas operaciones matemáticas muy comunes en el simulador (los sumadores), que no existen en la solución construida utilizando CLIPS. Una diferencia muy destacable es que en la solución de drools no se han asumido en ningún momento valores por defecto.

Respecto de drools-guvnor, la versión actual no se presenta de manera suficientemente sólida cómo para tratar con sencillez un problema de la complejidad que representa el simulador SIMBA. El problema de fondo es que no está suficientemente bien orientado para resolver conjuntos de reglas con tantas operaciones matemáticas cómo tiene el simulador. Para ser empleado en el contexto que se desea, sería necesario asumir que los usuarios que creen reglas van a tener conocimientos suficientemente amplios en drools. Actualmente drools-guvnor está mucho más orientado a resolver problemas sin tantas operaciones matemáticas.

Resumiendo, el problema se puede resolver perfectamente empleando drools como motor de reglas. Sin embargo las funcionalidades de drools-guvnor como BRMS no están lo suficientemente adaptadas a las necesidades concretas que no son un apoyo tan grande como sería deseable.

Respecto a aplicar una solución basada en reglas en vez de una solución basada en un lenguaje imperativo, actualmente se está volviendo a revisar parte de la formulación del rating de calidad, y hay que tener en cuenta que no es el único cambio que ha recibido este aplicativo desde que se inició el proyecto. Utilizar un enfoque basado en reglas hubiese simplificado enormemente la evolución del aplicativo. Además la utilización de las reglas de drools hubiese hecho posible que se involucraran personas con bajos conocimientos de programación, lo que hubiese mejorado la comunicación con las personas que mejor conocimiento tienen del problema.

Las principales ventajas que aporta drools (o un sistema de reglas) al simulador SIMBA:

- **Independencia del ciclo de desarrollo del resto de la aplicación**.
- **Acercar el problema a los expertos en simulación empresarial**.
- **Facilita y mejora la mantenibilidad del simulador**.

Como se puede apreciar en diferentes momentos del estudio, es de esperar que drools-guvnor siga creciendo como herramienta y algunas de las dificultades que se han encontrado vaya siendo subsanadas por el equipo de desarrollo.

7.2. Trabajos Futuros

Tras el trabajo realizado sobre el sistema SIMBA y el conocimiento adquirido en drools, se abren una serie de futuras líneas de trabajo. El origen de éstas líneas de trabajo viene sobre todo de los resultado del presente estudio, y cómo hacer triunfar la idea que éste estudio propone. Consciente de que el conocimiento sobre SIMBA es muy limitado, también se proponen algunas posibles mejoras sobre el sistema. Simplemente se lanzan las ideas para que las personas que han trabajado sobre la plataforma puedan decidir si son aportaciones interesantes o no.

Considerando que el resultado en la evaluación de drools-guvnor no ha sido el más deseado, sería interesante plantear la construcción de una herramienta propia que nos permita realizar las tareas que cubre, pero de una forma más ajustada a la solución del problema. Es decir centrar la construcción del BRMS en la situación ideal para su uso, el editor de reglas que mejor se ajustaría a las personas que lo van a usar, la definición de roles de usuario y privilegios que actualmente es más útil, el sistema de almacenamiento que mejor se ajusta a la plataforma física de la que se dispone, los mecanismos de comunicación más adecuados. Como peculiaridad sería interesante evaluar la posibilidad de construir un editor ajustado a las necesidades del problema y que sea independiente del motor de reglas

usado.

La opción de construir un BRMS propio puede suponer unos costes muy altos, por lo que probablemente se tendría que limitar su alcance, consiguiendo otro producto imperfecto. Parece mejor opción aplicar esfuerzos a mejorar las prestaciones de drools-guvnor. Al ser un software libre, y ya tener identificados los principales puntos donde necesita mejorar, es posible que con el esfuerzo adecuado se convierta en una herramienta que ofrezca todos los requisitos para solventar el problema.

Puesto que las herramientas que ofrece drools-guvnor no encuentran homólogos en CLIPS, y dado que por parte de drools se está realizando un proyecto para integrar las reglas que ejecuta el motor CLIPS en su propio motor de reglas, sería interesante tratar de trasladar todo el simulador construido en CLIPS a drools. Éste sería un estudio que asumiría mucho riesgo.

Cuando se describió SIMBA, se mencionó que justo antes de que la simulación se lance es posible que algún instructor introduzca la ejecución de un evento que afecte a la simulación. Sería interesante añadir componentes de lógica difusa en las reglas para discernir cómo sería el ranking de empresas afectado por la probabilidad de que ocurran los eventos que afecten a las simulaciones.

Drools como motor no ha ofrecido dudas sobre su viabilidad para resolver este tipo de problemas. Se podría abrir el debate sobre si una implementación basada en tecnologías JEE, mejoraría la actual arquitectura de SIMBA. Las tecnologías JEE están muy extendidas, esencialmente porque su uso mejora de manera considerable la productividad, especialmente en entornos web. Puesto que SIMBA consta de un importante peso de aplicación web se podría considerar si no sería más eficiente que en su totalidad se ejecute en un servidor de aplicaciones. Por supuesto habría que revisar si una implementación en JEE cumpliría todos los requisitos bajo los que el sistema fue diseñado.

Bibliografía

[1] Fernando Borrajo, Yolanda Bueno, Isidro de Pablo, Begoa Santos, Fernando Fernández, Javier García, Ismael Sagredo. *SIMBA: A simulator for business education and research*. http://e-archivo.uc3m.es/bitstream/10016/6825/1/simba_borrajo_DSS_2010_ps.pdf

[2] Asunción Gómez. *Ingeniería del conocimiento.*

[3] INTELIGENCIA ARTIFICIAL http://www.monografias.com/trabajos12/inteartf/inteartf2.shtml

[4] Charles L. Forgy, *Rete: A Fast Algorithm for the Many Pattern Many Object*

[5] *Página del Java Community process para definir el jsr 94.* http://www.jcp.org/en/jsr/detail?id=94

[6] Brett Stineman. *Answers to Top BRMS Questions*. ftp://public.dhe.ibm.com/common/ssi/ecm/en/wsw14085usen/WSW14085USEN.PDF

[7] Michal Bali. *Drools JBoss Rules 5.0 Developer's Guide*.

[8] *Página oficial de la documentación de drools*. http://www.jboss.org/drools/documentation

[9] *Página oficial de CLIPS, documentación*. http://clipsrules.sourceforge.net/OnlineDocs.html

[10] Daniel Sánchez Cisneros, Fernando Fernández. *Motor de razonamiento basado en reglas para el simulador empresarial SIMBA.*

[11] *Tracker de la versión 5.1 de drools-guvnor.* https://issues.jboss.org/browse/GUVNOR/fixforversion/12313201

[12] *Presentación de drools del Workshop de Nueva York de 2011.* http://www.slideshare.net/ge0ffrey/drools-new-york-city-workshop-2011

[13] *Manual de referencia de jee 5.* http://download.oracle.com/javaee/5/tutorial/doc/

[14] Barry Boehm, *A Spiral Model of Software Development and Enhancement.* http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/spiral.pdf

Apéndice A

Manual de Referencia

Éste es el manual de referencia del desarrollo llevado a cabo como apoyo al estudio realizado

A.1. Introducción

El desarrollo que se ha realizado es un pequeño desarrollo que se pueda emplear para realizar pruebas independientemente de la plataforma drools-guvnor. Así podremos llevar a cabo pruebas de integración, y test completamente independientes.

Para llevar a cabo el desarrollo se han utilizado las siguientes tecnologías (y sus correspondientes dependencias):

- **ant** http://ant.apache.org/: Es una librería y una herramienta, que permite ejecutar tareas de manera automática a partir de su descripción en un fichero. Algunas de las tareas más comunes en los proyectos que se automatizan con ant son: compilar, empaquetar, o ejecutar test.
- **Spring** http://www.springsource.org/: Es un framework empleado en el desarrollo de aplicaciones jee, que se caracteriza principalmente por: implementar el patrón de inyección de dependencias, por proveer elementos para abstraer los servicios empresariales que se usan en las aplicaciones y por facilitar el empleo de la programación orientada a aspectos.
- **XStream** http://xstream.codehaus.org/: Es una librería para la serialización sencilla de objectos Java en formato XML.
- **log4j** http://logging.apache.org/log4j/: Es una utilidad para tener log en las aplicaciones.

Las librerías que se han utilizado contienen numerosas dependencias, que emplean de manera interna pero que no han sido empleadas directamente durante el desarrollo involucrado.

Todas las librerías que se han empleado son opensource y por tanto se pueden emplear con total libertad en el presente proyecto.

La distribución incluye un fichero .war, donde se incluyen los ficheros binarios; y una carperta, donde se incluyen los fuentes del proyecto.

A.2. Instalación del proyecto

Existe dos elementos a instalar el fichero de extensión war y el proyecto con los fuentes. A continuación se detallan como llevar a cabo la implantación de los dos elementos.

A.2.1. Prerrequisitos

Para poder instalar el fichero war, será necesario la instalación de un servidor de aplicaciones compatible con la versión JDK 1.6 y la J2EE 1.4. Se recomienda el servidor de aplicaciones JBoss en la versión 4.2.3 (http://www.jboss.org/jbossas). Para poder instalar el servidor de aplicaciones es recomendable tener instalado el jdk de Java correspondiente. La compilación de los fuentes que se distribute se ha realizado con la versión 1.6.020 del jdk.

Además es necesario tener instalado drools-guvnor (http://www.jboss.org/drools/), existe la posibilidad de descargarse incluyendo la versión de JBoss 4.2.3.

Para poder trabajar con los fuentes es necesario tener instalado una versión del JDK 1.5 o superior. Es necesario tener correctamente instalada la herramienta ant. Es recomendable tener instalado el IDE Eclipse (http://eclipse.org), versión Gaminede o superior. La instalación de Eclipse suele instalar también ant.

Es importante que las herramientas se encuentren perfectamente instaladas, incluyendo las modificaciones pertienentes de las variables de entorno, en especial JAVA_HOME y ANT_HOME.

Todas las herramientas que se han mencionado en los prerrequisitos (tanto obligatorias como opcionales), son opensource.

A.2.2. Instalación del aplitivo war

Para poder ejecutar el aplicativo es necesario configurar en la máquina virtual de Java cierto elemento relacionado con Spring y AspectJ (librería relacionada con la programación orientada a aspectos). Lo que hay que configurar es que la máquina virtual arranque con un agente específico. Dicho agente está en el fichero org.springframework.instrument.jar que se encuentra en la carpeta lib que está dentro de la carpeta del proyecto. Se Debe copiar este fichero a un lugar accesible desde donde se arranque el servidor de aplicaciones y entre

las opciones con las que se arranca el servidor hay que incluir la opción -javaagent:<ruta donde está el archivo>/org.springframework.instrument.jar.

Una vez configurado el arranque del servidor, la instalación del programa depende de cuál sea el servidor de aplicaciones con el que estemos trabajando. En el caso del servidor de aplicaciones JBoss en la versión 4.2.3, es necesario copiar el archivo en la carpeta <ruta de la instalación de JBoss>/server/default/deploy.

Cuando se realice el despligue se puede comprobar si la instalación ha sido llevada a cabo con éxito accediendo a la página http://<nombre del host>:<puerto por el que escucha el servidor>/drools-integration, deberíamos ver la página que se muestra en la figura A.1.

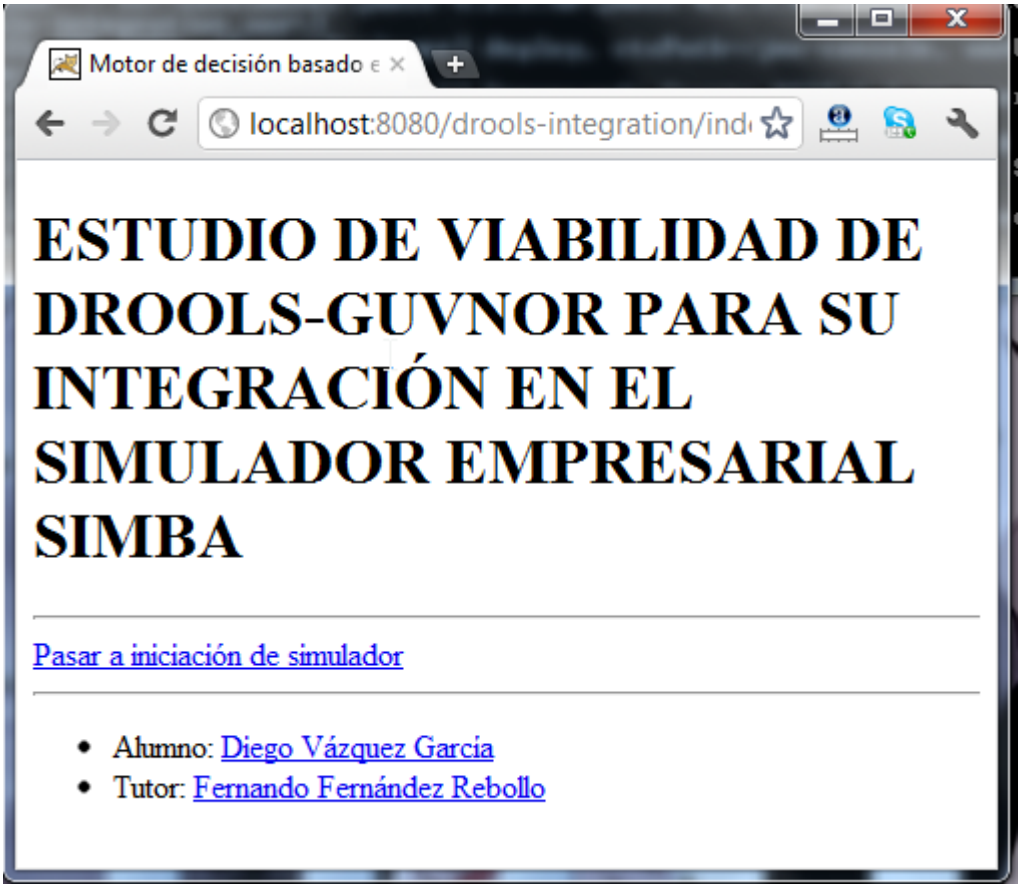


Figura A.1: Página de inicio.

A.2.3. Instalación del proyecto

Si el IDE Eclipse está instalado, simplemente debemos importar el proyecto. Para ellos copiaremos el proyecto en el workspace y crearemos un proyecto donde lo hayamos

instalado.

Si se escoge utilizar otro IDE diseñado especialmente para Java (como Netbeans) tenemos igualmente la opción de importar, aunque es posible que tengamos que configurar el classpath del proyecto.

Si bien es recomendable el uso de algún IDE para modificar el proyecto, no es imprescindible, ya que los ficheros de código fuente se pueden modificar con cualquier editor que escojamos y gracias a ant podemos relizar las tareas necesarias para desarrollar el proyecto.

Antes de empezar a trabajar con el proyecto seria conveniente configurar algunos parámetros de la herramienta de ant. Para ello en el fichero build.properties, que se encuentra en la raiz del proyecto, es necesario ajustar las propiedades al entorno donde se van a usar. Las propiedades a configurar son:

- **route_deploy**: La ruta donde el servidor de aplicaciones espera que se encuentre los fuentes de la aplicación.
- **server_cache**: El lugar donde el servidor de aplicaciones guardar los ficheros de caché de la aplicación.

A.3. Proyecto

A.3.1. Estructura del proyecto

En esta sección se va a detallar cómo esta estructurado el contenido del desarrollo dentro del proyecto A.2.

Los fuentes del aplicativo a desplegar en el servidor de aplicaciones se encuentran dentro de la carpeta **src**. Siguiendo la convención de nombrado de Java para paquetes todos los paquetes se llaman net.uc3m.droolsintegration (extensión inicial, nombre de la empresa o institución y nombre del proyecto).

Además existen otras dos carpetas con ficheros fuentes de java, que son **test** y **util**. En la carpeta test se guardan los test automáticos, y en la carpeta util un pequeño programa para compilar ficheros drl. La convención de nombrado que siguen es la misma que la carpeta src.

En la carpeta **lib**, se almacenan las librerías necesarias para alguna parte del proyecto, que no son necesarias en el aplicativo final.

El fichero de extensión war que se instala en un servidor de aplicaciones se guarda en la carpeta **dist**.

En la carpeta **testFiles** se guardan ficheros para realizar pruebas sobre las reglas de drools.

Todo el contenido del aplicativo necesario para ejecutarse en un servidor de aplicaciones se guarda en la carpeta **web**. Dentro de esta carpeta está la estructura de un apalicación jee. Las carpetas **css** y **tiles** son accesibles desde el path de la aplicación, una vez ubicada ésta en el servidor de aplicaciones. En la carpeta css se colocan las hojas de estilo, y en la carpeta tiles se encuentran los jsp. La carpeta más importante es la carpeta **WEB-INF**. El estándar jee define que esta carpeta debe existir dentro del path de la apliación y que además debe contener tres elementos como mínimo: el fichero descriptor de la aplicación **web.xml**, la carpeta con los fuentes de la aplicación **classes** y una carpeta con las librerías que requiere la aplicación **lib** en tiempo de ejecución (aunque puede que algunas también sean requeridas en tiempo de compilación). Ambas carpetas se cargan como parte del classpath de la aplicación.

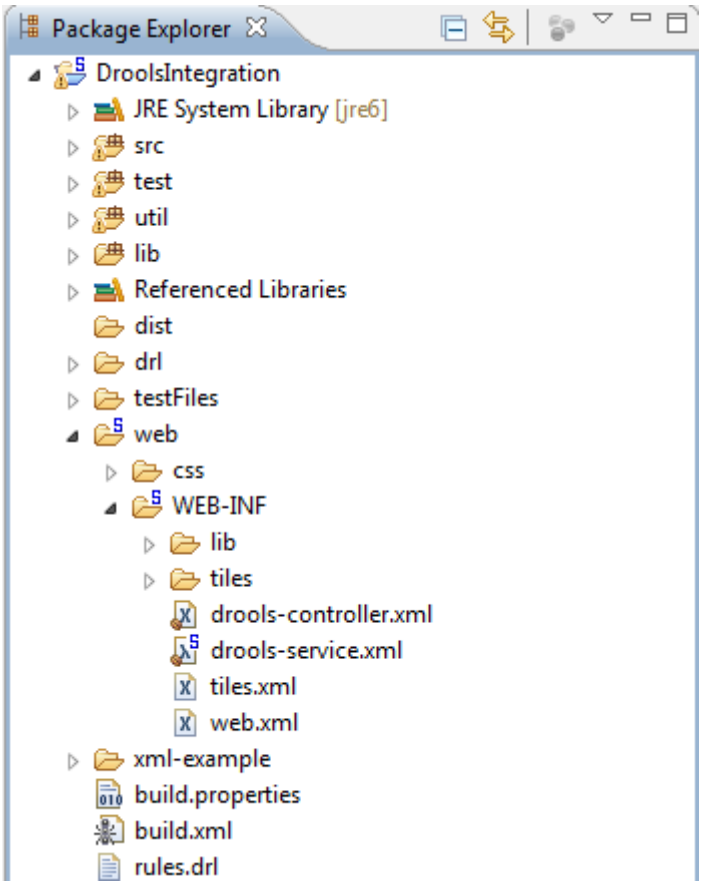


Figura A.2: Estructura del proyecto en el IDE Eclipse.

A.4. Funcionalidades

Para el estudio no era necesario construir un aplicativo, sin embargo para poder evaluar la integración y llevar a cabo cierta automaitzación en las pruebas ha sido necesario construir ésta pequeña aplicación. Su única funcionalidad es la posibilidad de relizar pruebas sobre un

conjunto de datos en formato XML y validarlo contra un conjunto de datos en el mismo formato. Para ello a través de un formulario se introduce la información necesaria para realizar la prueba (A.3):

1. **Fichero de hechos**: Es el fichero de hechos en formato XML que será el punto de partida para la ejecución del motor.
2. **Url**: Dirección del recurso donde se encuentra la base de reglas que se desea que ejecute el motor.
3. **Fichero con el resultado esperado**: Fichero con los datos que se espera que sea el resultado del motor tras terminar su ejecución.

Los datos de entrada se facilitan en formato XML. Se puede consultar los esquemas de los ficheros en los anexos correspondientes.

A.4.1. Líneas de ampliación

Existen dos elementos que se pueden modificar fácilmente dentro del proyecto, para poder ser reutilizado con otros propósitos. Uno de los puntos sería la fuente de datos y el otro el motor de reglas.

Para utilizar otra fuente de datos es necesarios construir una clase de envoltura sobre la fuente de datos, que debe implementar la interfaz FactLoader. Se recomienda leer el javadoc para entender las responsabilidades de los métodos necesarios. La idea es poder pasar los datos del modelo de datos de la fuente de información al modelo de datos que maneja la aplicación.

Si se desea probar otro motor de reglas, de igual forma que para cambiar la fuente de datos, se debería implementar la interfaz Engine. Sobre la clase de envoltura caerá la responsabilidad de traducir el modelo de datos que utiliza la aplicación al modelo de datos que requiere el motor. Por supuesto no sólo deberá añadir los hechos, también deberá cargar la base de reglas en el motor (a partir de una dirección de una fuente de datos) y ejecute el motor.

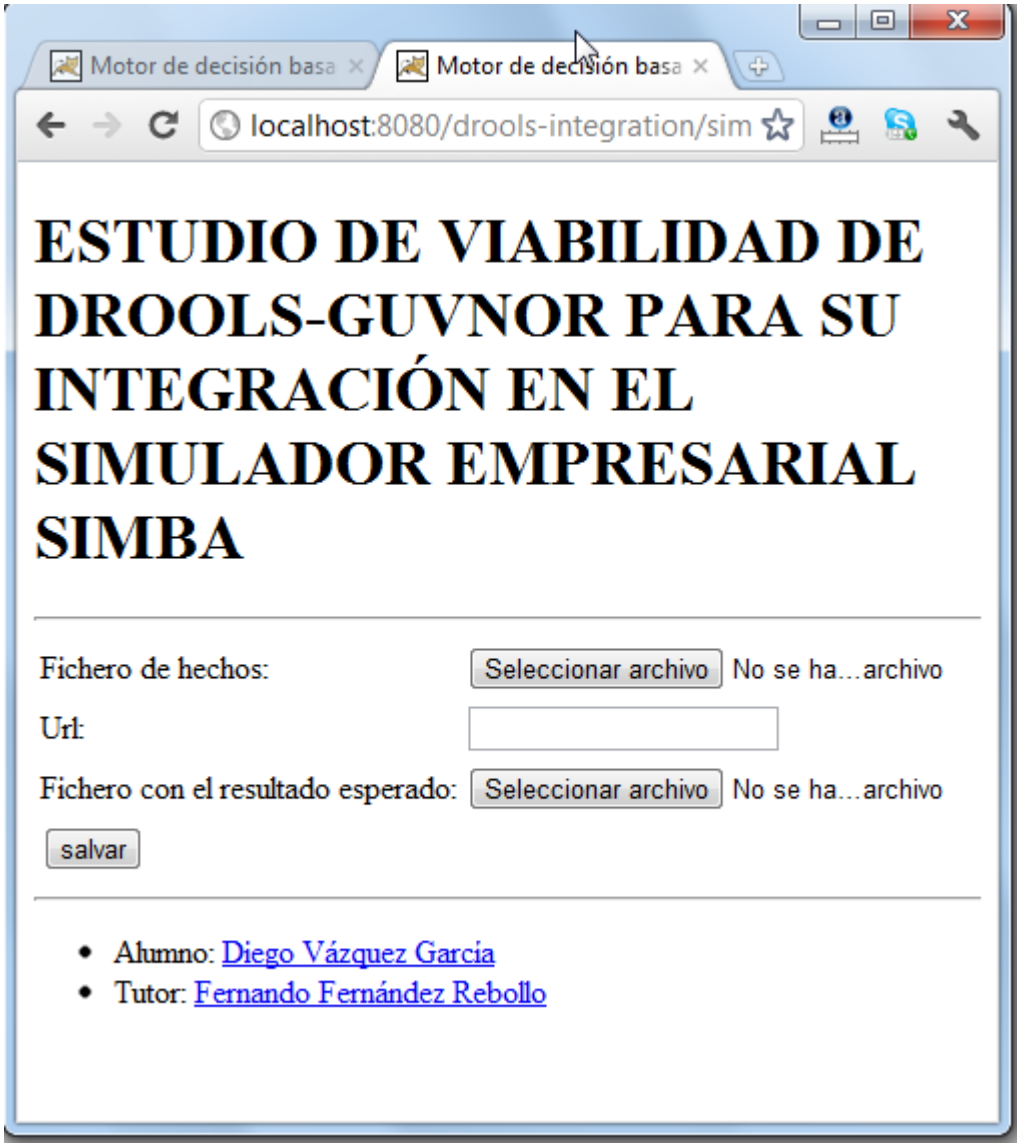


Figura A.3: Estructura del proyecto en el IDE Eclipse.

Apéndice B

Esquema de ficheros XML para la realización de pruebas

El sistema de pruebas desarrollado para testar el conjunto de reglas de una manera independiente, recibe dos ficheros XML. En uno de ellos se debe especificar los valores de inicio con los que empezar la simulación en otro el resultado esperado después de calcular todos los ratings de calidad. El fichero donde se define el esquema de los datos de inicio de la simulación está en la sección B.1 y el esquema para definir el XML del resultado en B.2.

B.1. Esquema 1: estado actual

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="facts">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="empresa"/>
        <xs:element maxOccurs="unbounded" ref="multiplicador"/>
        <xs:element maxOccurs="unbounded" ref="prefijo"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="empresa">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nombre"/>
        <xs:element ref="identificador"/>
        <xs:element ref="decisionesCalidadArray"/>
        <xs:element ref="segmentos"/>
      </xs:sequence>
    </xs:complexType>
```

```
</xs:element>
<xs:element name="identificador" type="xs:integer"/>
<xs:element name="decisionesCalidadArray">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="decisionesCalidad0"/>
         <xs:element ref="decisionesCalidad_1"/>
         <xs:element ref="decisionesCalidad_2"/>
         <xs:element ref="decisionesCalidad_3"/>
         <xs:element ref="decisionesCalidad_4"/>
         <xs:element ref="decisionesCalidad_5"/>
         <xs:element ref="decisionesCalidad_6"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="decisionesCalidad0">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="did"/>
         <xs:element ref="dpublicidad"/>
         <xs:element ref="dformacion"/>
          <xs:element ref="dprecemi"/>
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="dprecioemp"/>
            <xs:element ref="dpromred"/>
         </xs:choice>
         <xs:element ref="ratingCalidad"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="decisionesCalidad_1">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="did"/>
         <xs:element ref="dpublicidad"/>
         <xs:element ref="dformacion"/>
         <xs:element ref="dprecemi"/>
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="dprecioemp"/>
            <xs:element ref="dpromred"/>
         </xs:choice>
         <xs:element ref="ratingCalidad"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

```
<xs:element name="decisionesCalidad_2">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="did"/>
      <xs:element ref="dpublicidad"/>
      <xs:element ref="dformacion"/>
      <xs:element ref="dprecemi"/>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="dprecioemp"/>
        <xs:element ref="dpromred"/>
      </xs:choice>
      <xs:element ref="ratingCalidad"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="decisionesCalidad_3">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="did"/>
      <xs:element ref="dpublicidad"/>
      <xs:element ref="dformacion"/>
      <xs:element ref="dprecemi"/>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="dprecioemp"/>
        <xs:element ref="dpromred"/>
      </xs:choice>
      <xs:element ref="ratingCalidad"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="decisionesCalidad_4">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="did"/>
      <xs:element ref="dpublicidad"/>
      <xs:element ref="dformacion"/>
      <xs:element ref="dprecemi"/>
       <xs:choice maxOccurs="unbounded">
        <xs:element ref="dprecioemp"/>
        <xs:element ref="dpromred"/>
      </xs:choice>
      <xs:element ref="ratingCalidad"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="decisionesCalidad_5">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="did"/>
         <xs:element ref="dpublicidad"/>
         <xs:element ref="dformacion"/>
         <xs:element ref="dprecemi"/>
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="dprecioemp"/>
            <xs:element ref="dpromred"/>
         </xs:choice>
         <xs:element ref="ratingCalidad"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="decisionesCalidad_6">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="did"/>
         <xs:element ref="dpublicidad"/>
         <xs:element ref="dformacion"/>
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="dprecemi"/>
            <xs:element ref="dprecioemp"/>
            <xs:element ref="dpromred"/>
          </xs:choice>
         <xs:element ref="ratingCalidad"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="segmentos">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="segmento0"/>
         <xs:element ref="segmento_1"/>
         <xs:element ref="segmento_2"/>
         <xs:element ref="segmento_3"/>
         <xs:element ref="segmento_4"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="segmento0" type="xs:NCName"/>
<xs:element name="segmento_1" type="xs:NCName"/>
<xs:element name="segmento_2" type="xs:NCName"/>
<xs:element name="segmento_3" type="xs:NCName"/>
```

```
    <xs:element name="segmento_4" type="xs:NCName"/>
    <xs:element name="prefijo">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="tipoPrefijo"/>
                <xs:element ref="valor"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="tipoPrefijo" type="xs:string"/>
    <xs:element name="valor" type="xs:decimal"/>
    <xs:element name="nombre" type="xs:NCName"/>
    <xs:element name="did" type="xs:decimal"/>
    <xs:element name="dpublicidad" type="xs:decimal"/>
    <xs:element name="dformacion" type="xs:decimal"/>
    <xs:element name="dprecemi" type="xs:decimal"/>
    <xs:element name="dprecioemp" type="xs:decimal"/>
    <xs:element name="dpromred" type="xs:decimal"/>
    <xs:element name="ratingCalidad" type="xs:decimal"/>
    <xs:element name="multiplicador">
        <xs:complexType mixed="true">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="multiplicador"/>
                <xs:element ref="nombre"/>
                <xs:element ref="peridoRelativo"/>
                <xs:element ref="positivo"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="peridoRelativo" type="xs:integer"/>
    <xs:element name="positivo" type="xs:boolean"/>
</xs:schema>
```

B.2. Esquema 2: resultado esperado

```
    <?xml version="1.0" encoding="UTF-8"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="facts">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="condicionesMercado"/>
                <xs:element ref="empresa"/>
            </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>
  <xs:element name="condicionesMercado">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nombre"/>
        <xs:element ref="periodoRelativo"/>
        <xs:element ref="valor"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="periodoRelativo" type="xs:integer"/>
  <xs:element name="valor" type="xs:decimal"/>
  <xs:element name="empresa">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="identificador"/>
        <xs:element ref="ppptoscalidadi_dsegt_3"/>
        <xs:element ref="ppptoscalidadi_dsegt_2"/>
        <xs:element ref="ppptoscalidadi_dsegt_1"/>
        <xs:element ref="ppptoscalidadformat_3"/>
        <xs:element ref="ppptoscalidadformat_2"/>
        <xs:element ref="ppptoscalidadformat_1"/>
        <xs:element ref="ppptoscalidadpreciot_3"/>
        <xs:element ref="ppptoscalidadpreciot_2"/>
        <xs:element ref="ppptoscalidadpreciot_1"/>
        <xs:element ref="ppptoscalidadi_dt_3"/>
        <xs:element ref="ppptoscalidadi_dt_2"/>
        <xs:element ref="ppptoscalidadi_dt_1"/>
        <xs:element ref="ppptoscalidadredt_3"/>
        <xs:element ref="ppptoscalidadredt_2"/>
        <xs:element ref="ppptoscalidadredt_1"/>
        <xs:element ref="ppptoscalidadpublit_3"/>
        <xs:element ref="ppptoscalidadpublit_2"/>
        <xs:element ref="ppptoscalidadpublit_1"/>
        <xs:element ref="decisionesCalidadArray"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="identificador" type="xs:integer"/>
  <xs:element name="ppptoscalidadi_dsegt_3">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="multiplicador"/>
        <xs:element ref="peridoRelativo"/>
```

```
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadi_dsegt_2">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="multiplicador"/>
            <xs:element ref="peridoRelativo"/>
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadi_dsegt_1">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="multiplicador"/>
            <xs:element ref="peridoRelativo"/>
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadformat_3">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="multiplicador"/>
            <xs:element ref="peridoRelativo"/>
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadformat_2">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="multiplicador"/>
            <xs:element ref="peridoRelativo"/>
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
<xs:element name="ppptoscalidadformat_1">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="multiplicador"/>
      <xs:element ref="peridoRelativo"/>
      <xs:element ref="nombre"/>
      <xs:element ref="positivo"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadpreciot_3">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="multiplicador"/>
      <xs:element ref="peridoRelativo"/>
      <xs:element ref="nombre"/>
      <xs:element ref="positivo"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadpreciot_2">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="multiplicador"/>
      <xs:element ref="peridoRelativo"/>
      <xs:element ref="nombre"/>
      <xs:element ref="positivo"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadpreciot_1">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="multiplicador"/>
      <xs:element ref="peridoRelativo"/>
      <xs:element ref="nombre"/>
      <xs:element ref="positivo"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadi_dt_3">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="multiplicador"/>
      <xs:element ref="peridoRelativo"/>
```

```
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadi_dt_2">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="multiplicador"/>
            <xs:element ref="peridoRelativo"/>
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadi_dt_1">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="multiplicador"/>
            <xs:element ref="peridoRelativo"/>
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadredt_3">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="multiplicador"/>
            <xs:element ref="peridoRelativo"/>
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadredt_2">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="multiplicador"/>
            <xs:element ref="peridoRelativo"/>
            <xs:element ref="nombre"/>
            <xs:element ref="positivo"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
<xs:element name="ppptoscalidadredt_1">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="multiplicador"/>
      <xs:element ref="peridoRelativo"/>
      <xs:element ref="nombre"/>
      <xs:element ref="positivo"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadpublit_3">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="multiplicador"/>
      <xs:element ref="peridoRelativo"/>
      <xs:element ref="nombre"/>
      <xs:element ref="positivo"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadpublit_2">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="multiplicador"/>
      <xs:element ref="peridoRelativo"/>
      <xs:element ref="nombre"/>
      <xs:element ref="positivo"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ppptoscalidadpublit_1">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="multiplicador"/>
      <xs:element ref="peridoRelativo"/>
      <xs:element ref="nombre"/>
      <xs:element ref="positivo"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="decisionesCalidadArray">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="decisionesCalidad0"/>
      <xs:element ref="decisionesCalidad_1"/>
```

```
            <xs:element ref="decisionesCalidad_2"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="decisionesCalidad0">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="varID"/>
            <xs:element ref="varForm"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="decisionesCalidad_1">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="varID"/>
            <xs:element ref="varForm"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="decisionesCalidad_2">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="varID"/>
            <xs:element ref="varForm"/>
         </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="nombre" type="xs:NCName"/>
<xs:element name="multiplicador" type="xs:decimal"/>
<xs:element name="peridoRelativo" type="xs:integer"/>
<xs:element name="positivo" type="xs:boolean"/>
<xs:element name="varID" type="xs:decimal"/>
<xs:element name="varForm" type="xs:decimal"/>
</xs:schema>
```